Sistemas Informáticos
Curso 2003-2004

Intranet para un Dpto. Universitario

Daniel Fonseca Diaz
Mariano Herrera García
Gustavo Romero Benítez

Dirigido por:
Prof. Luis Hernández Yáñez
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Los miembros del grupo Daniel Fonseca Diaz, Mariano Herrera García y Gustavo Romero Benítez que han participado en el desarrollo del proyecto "*Intranet para un Dpto. Universitario*" dirigido por Luis Hernández Yánez autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Daniel Fonseca Diaz Mariano Herrera García Gustavo Romero Benítez

Intranet para un Dpto. Universitario

GUÍA DE LA DOCUMENTACIÓN

6 de Julio de 2004, v 2.0

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

1. PROLOGO

Este documento esta orientado a cualquier persona que quiera comenzar a leer la documentación del proyecto “Intranet SIP”.

Es un documento que sirve de guía y de índice del resto de la documentación, por lo tanto, debería ser leído en primer lugar y tenerse a mano mientras se leen los demás documentos.

2. INTRODUCCIÓN

El objetivo de este documento no es otro que ofrecer una guía rápida de cómo utilizar la documentación y un índice que sirva para localizar toda la documentación generada, además se incluyen impresiones personales a modo de conclusiones.

El documento esta estructurado en cuarto apartados:

- El primero explica como utilizar la documentación.
- El segundo recoge el índice de la documentación.
- El tercero es un breve resumen del método de trabajo que se ha seguido para desarrollar el proyecto.
- El cuarto son las conclusiones obtenidas del equipo que ha desarrollado el proyecto.

3. Uso de la documentación

Sólo se ha generado la documentación necesaria para llevar a buen fin el proyecto, eliminando documentos como gestión de riesgos, plan temporal... descritos en ingeniería del software y necesarios quizás para proyectos de cierta envergadura tanto en tiempo como en recursos pero no en este en concreto.

Siempre se ha pensado que la documentación debería ser útil para el desarrollo y para el mantenimiento, por este motivo toda la documentación tiene un marcado carácter técnico, sin embargo, se ha descartado la documentación excesivamente técnica haciendo referencia a la bibliografía mucho más profunda y a los comentarios del código.

No obstante se ha procurado exponer los problemas aparecidos durante el desarrollo sean o no de carácter técnico.

Desde el principio se ha pensado que es mucho mejor generar la documentación durante el desarrollo del proyecto y mantenerla en un proceso incremental al igual que el código. De esta forma se ha pretendido generar documentación de calidad actualizada y que sirva tanto para facilitar el desarrollo del proyecto como para su posterior explicación y comprensión.

Los documentos generados se han reunido en un único fichero en formato PDF por comodidad en la presentación, sin embargo, la idea original es utilizar este documento como guía para que el lector pueda acceder solo al material que le interese.

La división de la documentación en múltiples documentos esta encaminada a facilitar su mantenimiento, a medida que la Intranet amplié sus funcionalidades.

Al margen de que no se hallan respetado todas las normas de ingeniería del software, si que se han respetado los principios básicos del proceso unificado: Análisis, Diseño, Implementación, Prueba, Entrega...., de forma que se ha documentado cada una de estas fases de forma más o menos profunda.

Debemos recordar que forman parte de la documentación tanto la serie de documentos aquí descrita como los comentarios del código fuente...

En cualquier caso, si se quiere hacer una lectura secuencial de toda la documentación recomendamos seguir el índice expuesto en el siguiente apartado.

4. Índice de documentos y un pequeño resumen

El número que aparece entre paréntesis indica la cantidad aproximada de páginas del documento.

1. Informe Tecnológico: (40) Documento previo al desarrollo que explica la arquitectura aplicada y el diseño a grandes rasgos.
2. Capa de Controlador: (15) Expone el desarrollo de la capa del controlador.
3. Capa de Presentación: (40) Explica el desarrollo de la capa del controlador.
4. Capa Modelo (75)
    - Modelo de Datos: (30) Contiene el análisis y diseño del modelo de datos que se ha requerido.
    - EJBs: (12) Este documento contiene la explicación del grueso de la implementación de la capa del modelo.
    - Problemas de concurrencia: (15)
    - Optimización de Oracle: (10)
5. Gestión de Errores: (15) Se puede decir que es suplemento de la capa del controlador, explica como se han gestionado los errores en la aplicación,
    - Tratamiento de Excepciones: (15) En principio se creo como apéndice de ka Gestión de Errores, sin embargo durante el desarrollo se ha investigado bastante sobre este tema y ha terminado por conformarse como un documento independiente, recomendable para cualquier desarrollador.
6. Una Acción desde cero: (10) Creado con el fin de facilitar la implementación de nuevas funcionalidades.
7. Herramientas: Esta formado por una serie de documentos que exponen y explican algunas de las herramientas, utilidades y productos utilizados durante el desarrollo.
    - JBOSS (12): Es una guía práctica para instalar el servidor y configurarlo para poder desplegar la Intranet.
    - ANT (20): Es un resumen de cómo utilizar ANT para desarrollar aplicaciones web, utilizando este proyecto como ejemplo.
    - Log4J y CVS: Aunque en un principio se planifico documentar brevemente el uso que se ha hecho de estas herramientas, al final se ha desestimado ya que se puede encontrar mucha documentación sobre ambos y no se ha hecho ningún uso especial de estas herramientas.

5. Método de trabajo

Para desarrollar este proyecto se ha utilizado un proceso hecho a medida entre los típicos procesos increméntales por prototipos y los clásicos procesos en espiral.

No es el objetivo de este documento describir el modelo de proceso utilizado, ni se piensa que tenga especial importancia, sin embargo, si puede resultar interesante describir brevemente como a trabajado el grupo.

Se han utilizado distintas herramientas y recursos para sincronizar el trabajo:

- Acuerdos de arquitectura y análisis: Después de estudios individuales y mediante correo electrónico y breves reuniones se han ido acordando las decisiones de más alto nivel.
- Diseño: El responsable asignado ha pensado el diseño de cada módulo y documentado antes de llevarlo a cabo, de forma que cada miembro del grupo ha estado enterado en todo momento de cómo se ha construido cada parte y con la suficiente antelación como para poder aportar mejoras al diseño.
- Implementación: Entendiendo como implementación la generación de código fuente y el cierre de la documentación correspondiente. Se ha realizado de la siguiente forma.
    i. Cada responsable a comenzado la implementación de su parte.
    ii. El resto de miembros la han repasado y corregido los bugs encontrados.
    iii. El responsable a completado la documentación.
    iv. El resto de miembros la han repasado.
    v. Después del cierre de cada bloque se ha realizado una reunión para acordar que esta cerrado y la documentación ha sido remitida al tutor del proyecto para ser revisada antes de la entrega final.

Para llevar a cabo todo esto se han utilizado las siguientes herramientas y recursos.

- Cada cual ha utilizado las fuentes a su disposición para investigar y estudiar las distintas alternativas: Páginas web, foros, biblioteca...
- Plasmar todo el diseño en UML ha parecido excesivo y poco útil, por lo que se han utilizado documentos en formato electrónico (WORD, PDF).
- La sincronización de la documentación (liberación y revisión) se llevo a cabo en un primer momento mediante un grupo de trabajo en Internet, en concreto los ofrecidos por Yahoo **[YAHGR]** y después un módulo del CVS.
- La sincronización del código se ha llevado a cabo con un servidor CVS **[CVS]** alojado en un servidor facilitado por el departamento.
- Para realizar las pruebas y depuración se ha utilizado el servidor facilitado por el departamento, donde se instalo: JAVA, JBOSS, ORACLE, ANT y CVS.

6. Conclusión

6.1 Objetivos cubiertos

El proyecto fue planteado como el diseño de una aplicación WEB que sirviese como base para el desarrollo de Intranet para Departamentos Universitarios.

Este tipo de Intranet se caracterizan por:

- Necesitar seguridad.
- Diseño relativamente estable.
- Manejar un pequeño número de usuarios.
- Manejar gran cantidad de documentos estáticos.
- Capacidad de modificar datos dinámicamente (fechas, notas...)
- Necesitar nuevas funcionalidades cada poco tiempo (semestres)

Se ha diseñado un marco de trabajo de buena calidad, que soporta estos requisitos, de forma que se pueda trabajar sobre él y ampliarlo para conseguir las funcionalidades de cualquier Intranet.

El sistema es portable basado en software libre y tecnologías JAVA y XML.

El sistema esta internacionalizado, aunque en principio esta implementado en Ingles y Español, no hay mayor problema en ampliarlo a otros idiomas, sin necesidad de tocar el código de la aplicación. No obstante incorporar idiomas como el Alemán puede resultar complicado ya que el tamaño medio de las palabras puede obligar a modificar el diseño.

Se han desarrollado un sistema para permitir el despliegue automático de la aplicación, basado en tareas de ANT.

Se pueden generar trazas de debug, error... de forma sencilla y altamente configurable, basado en LOG4J.

Se ha trabajado profundamente en la gestión de errores.

6.2 Objetivos no cubiertos

Los objetivos no cubiertos se pueden dividir en dos grupos:

Documentación:

Han quedado pendientes documentos importantes:

- Instalación
- FAQ
- Trucos y consejos

Y otros documentos no tan importantes como:

- Uso de CVS
- Guía de Log4J
- Compendio de herramientas

En líneas generales la documentación esta completa, aunque no se ha conseguido plasmar toda el trabajo de investigación que se ha llevado a cabo.

Diseño e Implementación:

- Integración completa de JAAS en el módulo de Login.
- Automatizar el proceso de instalación.
- Generar una herramienta para migrar la anterior Intranet.
- Implementar todas las funcionalidades analizadas.
- Implementar por completo el sistema para gestionar errores.

6.3 Que hemos aprendido

Gracias a la labor de investigación que se ha llevado a cabo, los miembros del equipo han adquirido obviamente conocimientos sobre nuevas tecnologías... y por otro lado e igualmente importante, cada uno ha mejorado la forma de trabajo tanto personal como en grupo.

Dentro de las tecnologías aprendidas destacan:

- Diseño con Patrones, que no se habían estudiado en otras asignaturas como Ingeniería del Software.
- Profundización en el API de Java: JDBC, Reflection, Servlets, Asserts...
- Se han asentado conocimientos sobre Servidores de Aplicaciones Web (servidor de páginas, motor de servlets, contenedor de EJBs).
- Se ha investigado en temas relativos a entornos servidor (securización, acceso y gestión remota).
- Se ha investigado en teorías de excepciones.
- Se han aprendido a utilizar herramientas de última generación (IntelliJ Idea, Toad, XML Spy...)
- Se han adquirido conocimientos técnicos sobre Oracle y asentado conocimientos de bases de datos en general.

Dentro del apartado no-técnico podemos destacar tres temas:

- Ligados a la organización personal y del equipo, la utilización de un servidor CVS para gestionar las versiones de código y documentación ha sido muy valiosa en distintos ámbitos de cada uno.
- La utilización de software libre nos ha permitido empezar a comprender esa filosofía de trabajo y participar en ella, resolviendo fallos en software en el que participan miles de programadores.
- Adentrarnos en tecnologías novedosas ha permitido al equipo aprender algo muy importante: Aprender a investigar.

6.4 Opinión personal

Aunque en un principio nos hemos quedado cortos en cuanto a funcionalidad se ha conseguido un software de alta calidad, respetuoso con las reglas de la programación moderna desacoplado, modularizado, fácilmente reutilizable y eficiente.

En el apartado de la documentación, que se ha considerado tan importante como el software, creemos que esta muy bien aunque no sea excesivamente formal, ni se halla profundizado tanto como nos hubiese gustado.

En resumen, reflexionando sobre este apartado y volviendo atrás en el tiempo podemos concluir que después de la realización del proyecto no solo hemos ampliado nuestros conocimientos técnicos, sino lo que es más importantes nos hemos empezado a convertir en buenos profesionales responsables de nuestras tareas asignadas, capaces de trabajar en equipo sincronizadamente y sobre todo capaces de analizar, diseñar e implementar software de calidad en tiempos razonables.

Otra conclusión importante más orientada al aspecto tecnológico del proyecto es que después de leer abundante documentación sobre J2EE, XML, Oracle, patrones... creemos que es cierto que se obtiene software que cumple todos los principios de la programación orientada a objetos, siendo fácil de mantener (depurar y ampliar) pero muy costoso de construir y puede llegar a ser terriblemente complicado lo que implica la necesidad de expertos en cada tecnología.

Una de las grandes ventajas de la arquitectura propuesta J2EE... es que es rentable un futuro próximo ya que aunque cuesta más generar el software, después es más fácil de mantener... sin embargo, todas estas ventajas se ven superadas por el hecho de que es una tecnología demasiado dinámica, completamente inestable en el tiempo, por ejemplo:

- Aún no se ha formalizado la forma de trabajar con EJBs cuando, existe una nueva especificación EJB 2.0
- Cuando los desarrollos que utilizan JDBC comienzan a ser rentables, surgen la tecnología JDO.
- Los JSP y Servlet son sustituidos por los JSLT.
- Frameworks estables como Struts son desechados en favor de otros como Spring.

Y así podemos continuar con una larga lista de ejemplos.

CONTROL DE VERSIONES

Este documento no esta cerrado y se irá ampliando según sea necesario hasta finalizar el proyecto.

1.0 → Estructura y resumen

1.1 → Índice

2.0 → Conclusiones

INFORME TECNOLÓGICO

Intranet para un Dpto. Universitario

ARQUITECTURA Y TECNOLOGÍA UTILIZADAS

15 de Diciembre de 2003, v 2.0

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

1. ARQUITECTURA TÉCNICA

1.1 Introducción

Esta sección define todos los componentes que forman parte de la solución del proyecto *IDSIP*[1]. Todos ellos están basados en las especificaciones J2EE de Sun microsystems como se decidió en el acuerdo del proyecto.

La sección de Arquitectura Técnica se divide en dos partes:

- La primera trata cuestiones teóricas sobre aplicaciones cliente-servidor clásicas. Para comprender mejor la solución propuesta y sus ventajas, se explica cómo se estructuran ciertas arquitecturas y su evolución. Primero se describen las soluciones basadas en arquitecturas de tres capas y cuáles son las características generales de los sistemas que las implementan. Luego cómo maneja estos conceptos la tecnología J2EE y finalmente se presenta un marco de trabajo para desarrollar según la especificación J2EE, sobre la arquitectura en tres capas.
- La segunda se centra en la solución adoptada para la realización del proyecto dando paso al informe sobre las tecnologías utilizadas.

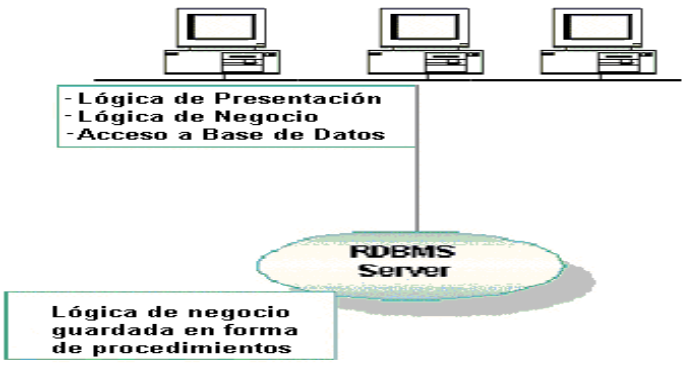
1.2 Presentación de los sistemas de tres capas

1.2.1 Objetivo

Este apartado presenta la arquitectura de tres capas en forma general, introduciendo las características comunes a distintas especificaciones como J2EE.

1.2.2 Antecedentes (Cliente-Servidor 2 capas)

El antecedente inmediato de la arquitectura de tres capas, es la arquitectura de dos capas. Cuenta con una estación de trabajo como cliente (*front-end*) y un servidor de bases de datos que contiene la información (*back-end*). Las reglas del negocio están repartidas entre el programa cliente y procedimientos residentes en el SGBD[2], como se muestra en la siguiente figura:



[1] A falta de darle un nombre al proyecto. IDSIP: intranet del Departamento de Sistemas Informáticos y Programación de la Universidad Complutense de Madrid.

[2] SGBDR (***RDBMS***): Sistema Gestor de Bases de Datos Relaciónales (Oracle, SQL Server...).

Este modelo funciona bien si existe comunicación rápida y fiable entre los componentes y solamente hay una fuente de datos.

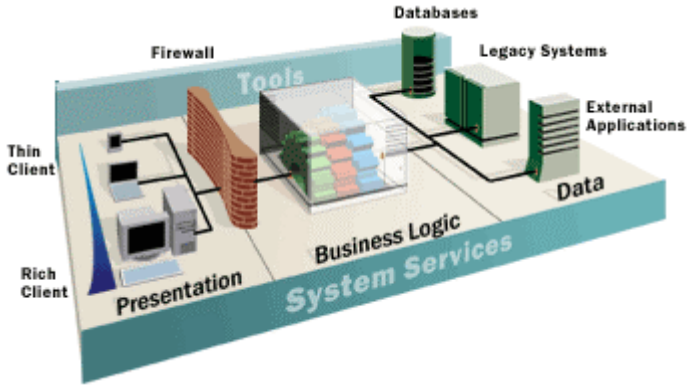
Las principales desventajas son las siguientes:

- Cuando se modifican el comportamiento del negocio y hay que distribuir las nuevas versiones de las aplicaciones en todos los clientes. Problemas de versiones de cada cliente...

- Cuando se quiere hacer interactuar a varias aplicaciones construidas sobre esta arquitectura, pero usando proveedores de base de datos diferentes. Las reglas de negocios de cada base, deberán ser replicadas en las otras. La programación de las reglas de negocio, tendrá que incluir la coordinación de transacciones entre distintas bases de datos. Deberán distribuirse entre los clientes, los drivers para cada una de las bases de datos con las que deban comunicarse...

1.2.3 Presentación de la arquitectura de tres capas

El hecho de que fuera difícil escalar las aplicaciones cliente-servidor, dio lugar a la división de las aplicaciones distribuidas en tres capas. Esta división se representa en la siguiente figura.



Cada una de estas capas, realiza un tipo distinto de procesamiento:

- Servicios de usuario o capa de presentación.
- Servicios de negocio o capa de dominio.
- Servicios de datos o capa de acceso a datos.

No hay que confundir estas tres capas con el concepto *Modelo Vista Controlador,* que se comentará más adelante.

También se habla de arquitecturas de ***n*** *capas*, esto se refiere a las distintas subdivisiones de las tres capas principales.

Con esta organización se reduce la complejidad de la aplicación y se mejora su escalabilidad y mantenibilidad, ya que:

- Se fomenta la separación de roles, de forma que los desarrolladores pueden especializarse en una determinada capa, aumentando su productividad.

- Se fomenta el empleo de herramientas especializadas en diseño, desarrollo, acceso a bases de datos...

1.3 Especificación J2EE (Java 2 Enterprise Edition)

1.3.1 Introducción

La actual necesidad de desarrollar aplicaciones distribuidas que trabajen en forma transaccional, conservando niveles de rapidez, seguridad y escalabilidad, han ido aumentando.

La estructura tradicional de dos capas (cliente/servidor) ha ido evolucionando en estructuras más complejas, formadas por más capas, en las cuales se busca una división clara de los roles que forman parte de las componentes de las aplicaciones, contribuyendo a una mejor reutilización, crecimiento y mantenibilidad de los sistemas existentes.

Bajo este contexto se creó ***Java 2 Enterprise Edition*** (J2EE), una plataforma creada por Sun Microsystem y basada en Java, para el desarrollo de aplicaciones empresariales distribuidas. J2EE se basa en una estructura multicapa para el desarrollo de dichos sistemas.

La división de capas que se propone es la siguiente:

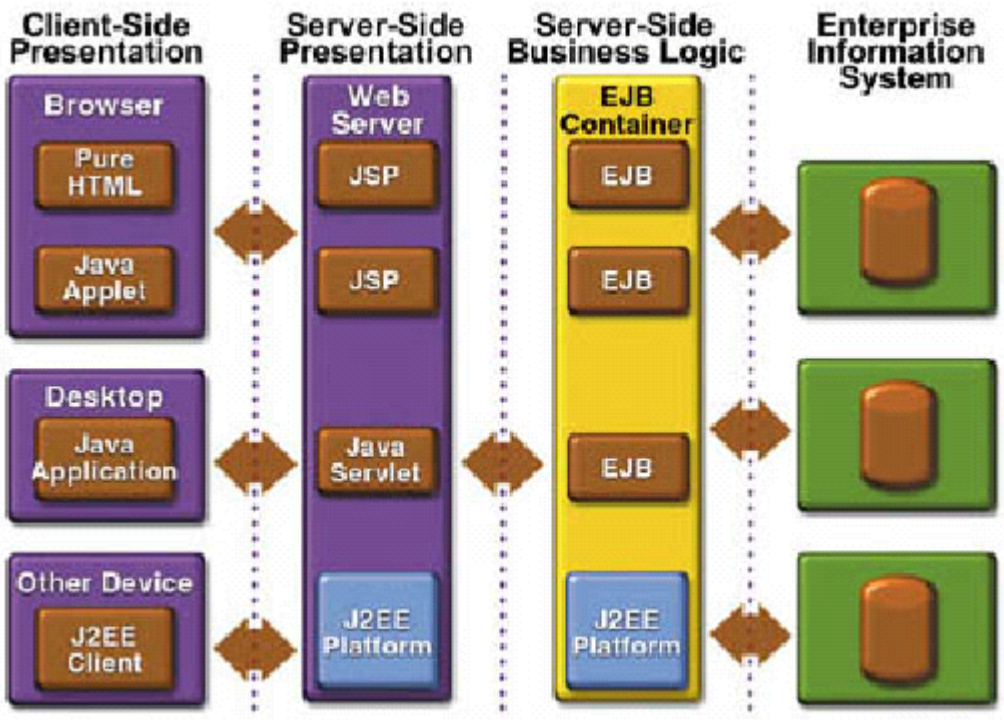
Nombre	Quiénes la componen	Dónde se ubica
Capa Cliente	App. cliente, applets…	PC Cliente
Capa de Presentación	JSP, Servlet y otras Uis	Servidor J2EE
Capa de Negocios	EJB y objetos de negocio	Servidor J2EE
Capa de Integración	JMS, JDBC	Servidor J2EE
Capa de Recursos BD	Datos (tablas...)	SGBD

J2EE abarca las Capas de Presentación, de Negocio y de Integración, esto a través de diversos servicios y contenedores de objetos que facilitan su publicación e interacción con las otras capas.

J2EE es entonces una especificación que se acopla sobre la arquitectura de tres capas, es una plataforma que proporciona los medios para diseñar software basado en una arquitectura de tres capas; además da unas recomendaciones de diseño utilizando patrones software.

1.3.2 Conceptos sobre Aplicaciones Web Distribuidas

J2EE esta diseñada para soportar el concepto *DTP*, procesamiento de transacciones distribuidas. La idea fundamental de este concepto es que una transacción de negocio, puede ser ejecutada expandiéndose por servidores y sistemas distribuidos y heterogéneos de forma que se pueda escalar fácilmente tanto verticalmente o como horizontalmente a un nivel lógico y físico.



La aplicación deberá contar con un Contenedor[3] que cumpla la especificación J2EE. Las tecnologías que forman J2EE fueron desarrolladas por Sun Microsystems y otros proveedores de software libre y comercial conjuntamente.

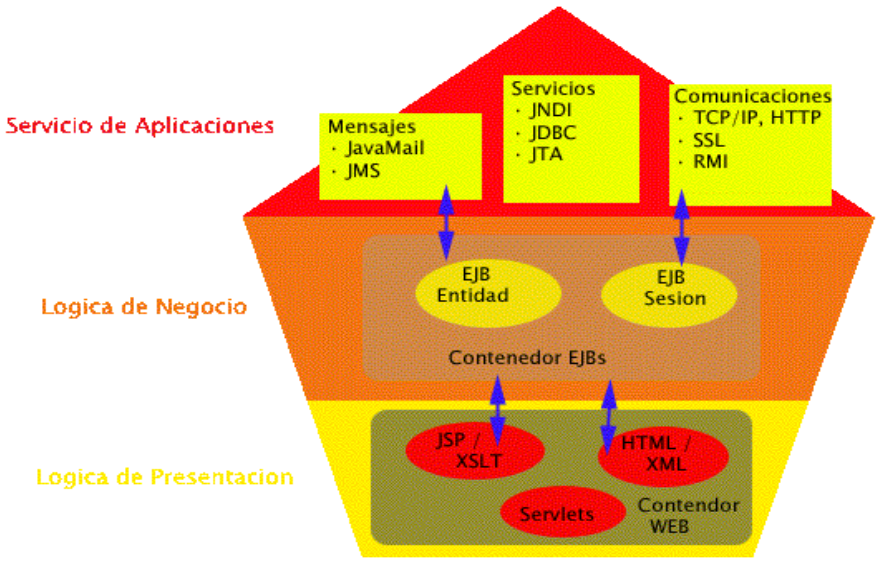
Este servidor será utilizado como una plataforma de desarrollo y ejecución de las aplicaciones basadas en componentes. Dentro de la arquitectura, será el responsable de la ejecución del middleware, es decir el Contenedor WEB es a una aplicación WEB lo que un sistema operativo a un aplicación personal, entre otras cosas proporciona el marco transaccional necesario para el aseguramiento de la consistencia de los datos, gestiona las peticiones del cliente y el envío de las respuestas, maneja la memoria de aplicación, sesión y petición de cada cliente, proporciona un servicio para la localización de objetos distribuidos...

[3] Ver Apéndice 1, sobre Servidores de Aplicaciones, Java Engines y contenedores EJB

1.3.3 Componentes de la plataforma

El servidor provee un conjunto de servicios, estos servicios incluirán soporte para Servlets, Java Server Pages (JSP), y Enterprise JavaBeans (EJB).

Los servicios incluyen el acceso a los protocolos de red estándares, acceso a sistemas de bases de datos (JDBC), y sistemas de mensajería (JMS). Los componentes de la aplicación son ejecutados por el contenedor Web. Estos contenedores brindan el soporte para el ciclo de vida y los servicios definidos por las especificaciones J2EE por lo que los componentes no tienen que manejar estos detalles.



A continuación se realiza un desglose de los principales elementos y tecnologías de cada capa. El objetivo de este apartado no es explicar cada tecnología sino comentar que aporta cada tecnología y para que es útil; es en la siguiente sección donde se da un repaso general de cada una de estas tecnologías y otras que también se utilizan en el proyecto en el que se incluye referencias a documentación mas extensa.

1.3.4 Capa de presentación (Servicios de Usuario)

Las aplicaciones Web son, por la naturaleza de su tecnología, fáciles de acceder y portar. En una aplicación Web, la interfaz del usuario está representada normalmente por páginas HTML estático o generado dinámicamente por *JSPs*, *taglibs* y *servlets*. El navegador (browser) contiene la lógica para interpretar y desplegar la página de acuerdo con la especificación de la página HTML.

Un servlet, recibe los parámetros de una petición HTTP en un objeto ***request*** y, típicamente escribe HTML o XML en su objeto ***response***. Las páginas JSP son convertidas a servlets antes de ser ejecutadas en el servidor de aplicaciones, por lo que JSP y Servlets son diferentes representaciones de lo mismo. JSP son distribuidas de la misma forma en que se distribuyen páginas HTML, el servidor Web debe tener acceso a los .jsp, al igual que a los .html. Cuando un cliente solicita un archivo .jsp, el servidor verifica si se ha compilado o si el archivo ha cambiado desde la última compilación, y si es necesario llama al compilador JSP del servidor de aplicaciones, el cual genera el código correspondiente a un servlet Java para luego ser compilado a un archivo Java .class.

1.3.5 Capa de lógica (Servicios de Negocio)

Enterprise JavaBeans son los componentes donde reside la lógica de negocios para una aplicación J2EE, estan alojados en el servidor de aplicaciones, el cual gestiona su ciclo de vida y otros servicios como cacheado, persistencia, y manejo de transacciones.

Aunque los distintos tipos de EJB se detallan en la sección de tecnología, se introducen ahora para poder explicar la solución adoptada.

- **Message-Driven Beans**

Message-driven beans, introducidos en la especificación EJB 2.0 manejan mensajes asincrónicos recibidos desde colas de mensajes JMS.

- **Session Beans**

Los beans de session tienden a implementar **lógica del negocio**. Una instancia de un S-EJB es una instancia transitoria que sirve a un cliente; están destinados a enfrentar acciones mas que datos. El contenedor EJB crea un bean de sesion al ser requerido por el cliente. Luego es mantenido mientras el cliente mantenga su conexión con el bean. Aunque **no** son persistentes, pueden salvar datos a un almacenamiento permanente si lo requieren.

Si estos beans no mantienen un estado específico con el cliente entre sucesivas llamadas, y puede ser utilizado por cualquier cliente se denominan *Stateless* (sin estado). Pueden ser usados para ofrecer acceso a servicios que no dependen del contexto de una sesión.

Si mantiene el estado en nombre de un cliente específico, se denominan *Stateful* (con estado). Estos beans pueden ser usados para manejar procesos. Como pueden acumular y mantener el estado a través de múltiples interacciones con el cliente, son a menudo los objetos controladores en una aplicación.

Como no son persistentes, los S-EJBs deben completar su trabajo en una sola sesión, por medio del uso de JDBC, JMS, o E-EJBs para guardar su trabajo de forma permanente.

- **Entity Bean**

Los bean de entidad contienen **Datos** y métodos para acceder a ellos. Las instancias de los E-EJB representan datos persistentes de la aplicación.

Estos bean suelen estar mapeados con objetos del SGBD (tablas, filas, columnas) generalmente por medio de JDBC.

La persistencia de estos bean puede ser manejada por el contenedor o por el propio bean, dando lugar a dos subtipos de bean de entidad: *CMP* y *BMP*

1.3.6 Capa de datos (Servicios de Datos)

Esta capa se engloban los datos almacenados y los métodos de acceso a los mismo, se encapsula así su diseño, residencia física y reglas de negocio aplicables. Los métodos deberían ser **reusables**, y deben garantizar la integridad sintáctica y semántica de los datos que manipulan.

Esta suele subdividirse en tres sub-capas con funcionalidades específicas:

- Métodos de acceso a datos de alto nivel: son los componentes responsables de la manipulación de los datos de acuerdo a las reglas del negocio específicas a los mismos, que garantizan el cumplimiento de aquellas validaciones de integridad no implementables por mecanismos estándar en la propia base de datos (PK, FK...). Esta capa preferentemente residirá físicamente en el Servidor de Aplicaciones.
- Métodos de acceso a datos de bajo nivel: se incluye en esta capa las sentencias de acceso a los datos (select, insert, update, delete, invocación de pl ...) en SQL del propio gestor de base de datos, por lo tanto, podría sufrir modificaciones si se reemplaza el motor de base de datos, de hecho esta subcapa puede contener las invocaciones al middleware de acceso a la base.
- Base de datos: además de los datos físicos, también contiene aquellas reglas del negocio que garantizan la integridad de los datos y que son implementables por mecanismos propios de la base PK, FK, triggers... sean cuales sean estos mecanismos deben ser estándares o fácilmente migrables a otros SGBD.

APIs de acceso a datos y servicios

Las APIs que ofrece la especificación J2EE para acceder a datos son las siguientes:

- **JNDI**

Java Naming and Directory Interface es una API java estándar que permite a las aplicaciones a buscar objetos distribuidos por su nombre.

Una aplicación puede buscar un objetos como JDBC DataSources, EJB, RMI, colas y tópicos JMS y por medio de un contexto JNDI, para luego invocar los métodos de búsqueda JNDI con el nombre del objeto.

Tradicionalmente, se utilizaban distintas APIs para acceder a distintos servicios de directorio, como LDAP o NIS pero JNDI se puede acceder a cualquier tipo de directorio.

- **JDBC**

Java Database Connectivity proporciona el acceso a los recursos backend de bases de datos. Las aplicaciones Java necesitan un driver JDBC, que hace las veces de interfaz para una base de datos específica como Oracle.

- **JTA**

Java Transaction API es la interfaz estándar para manejar transacciones en aplicaciones Java. Por medio del uso de transacciones, se protege la integridad de los datos en la base de datos y gerencia el acceso a los datos por aplicaciones concurrentes o instancias de aplicaciones. Una vez que una transacción comienza, toda operación transaccional que se confirme exitosa debe hacer commit o todas las operaciones deben se deshechas haciendo rollback.

1.4 Patrones de Diseño

1.4.1 Introducción

Este apartado no corresponde a un informe tecnológico, sino más bien a un glosario. Sin embargo, consideramos que es muy importante introducir el concepto de patrón para la perfecta comprensión del documento, por lo tanto, si el lector esta familiarizado con este concepto puede pasar al siguiente punto.

1.4.2 Definición

Un patrón de diseño describe un problema que ocurre repetidas veces en algún contexto determinado de desarrollo de software y presenta una buena solución ya probada, a menudo esta solución es modelada mediante UML.

Esto ayuda a diseñar correctamente en menos tiempo, ayuda a construir problemas reutilizables y extendibles, facilita la documentación, y facilita la comunicación entre los miembros del equipo de desarrollo.

1.4.3 Patrones de Diseño J2EE

Un grupo de investigadores que forman parte de Sun, desarrollaron un catálogo[4] de patrones para ser utilizados en el diseño de aplicaciones basadas en J2EE. El catálogo completo puede ser encontrado en el libro *"Core J2EE Patterns book by Deepak Alur, John Crupi, and Dan Malks"*.

Están clasificados según su funcionalidad dentro del modelo de n capas que se ha comentado anteriormente, esto es:

- Patrones de la capa de presentación
- Patrones de la capa de negocios
- Patrones de la capa de integración

Cabe señalar que todos estos patrones poseen tanto características de diseño como de arquitectura. En el apéndice 2 podemos encontrar un resumen de los patrones más utilizados de este cátalogo.

[4] Catálogo que puede encontrarse en http://java.sun.com/blueprints/corej2eepatterns/index.html

1.5 Aplicación de J2EE a aplicaciones Web

1.5.1 Objetivo

Este apartado estudia como mediante la aplicación de tecnologías de objetos distribuidos, especificadas en J2EE, se consigue cumplir o alcanzar algunos de los objetivos marcados por la arquitectura de tres capas.

1.5.2 Separación lógico-física de capas

Suponiendo una arquitectura de tres capas, el primer paso a la hora de emprender el diseño de una aplicación web, es conseguir separar conceptualmente las tareas que el sistema debe desempeñar entre las distintas capas lógica, diferenciando que procesos responde a tareas de presentación, cual a negocio y cual a acceso a datos. En caso de identificar algún proceso lógico que abarque responsabilidades adjudicadas a dos o más capas distintas, dicho proceso debe ser dividido en subprocesos, hasta alcanzar el punto en el que no exista ninguno que abarque más de una capa lógica.

1.5.3 Separación física de las capas

Se analiza la separación física de las responsabilidades, para obtener una aplicación escalable verticalmente.

La evolución del modelo 3-capas al modelo n-capas pasa por la incorporación de las capas intermedias que permiten desacoplar las primitivas y distribuirlas por medio de algún tipo de middleware. Así, cada una de las capas deberá ser vista como un conjunto de servicios desde la capa superior, encapsulando la lógica que implementen. A continuación se expone una solución basada en objetos distribuidos como los EJB.

EJBs de tipo Sesión

El principio de separación lógica y física de capas implica que los procesos de negocio son implementados en una capa superior a la de persistencia. Es habitual, sobre todo en los proyectos de comercio electrónico –carrito de la compra...-, que en la capa de negocio aparezca la necesidad de realizar operaciones de persistencia de entidades de carácter atómico, en las cuales la ejecución parcial de las mismas lleva a la base de datos a un estado de inconsistencia. La forma de conseguir que las operaciones de la capa de negocio puedan estar dotadas de este carácter es el empleo de transacciones distribuidas. Los EJBs están dotados de esta característica. El contenedor de EJBs que cumpla con la especificación J2EE controla las transacciones distribuidas en las que participan los componentes que gestiona. Se deduce así como la más recomendable de las tres la solución basada en el empleo de EJBs de sesión como implementación de las capas intermedias (middleware).

Bajo este prisma, los EJBs actúan como objetos tontos, carentes de lógica alguna, cuyo cometido único es hacer de puente entre la capa superior y la inferior, ocultando la lógica que ejecutan los métodos que publica. Este funcionamiento responde al del patrón de diseño Fachada (*Facade Pattern*).

De esta forma, haciendo que EJBs de sesión muestren a la capa superior el conjunto de servicios que ofrece la capa inferior, al mismo tiempo que posibilita la invocación remota de los mismos de forma transparente al usuario del servicio se consigue que la separación

lógica entre capas pasa a ser también física, pudiendo dotar así a la aplicación de escalabilidad vertical.

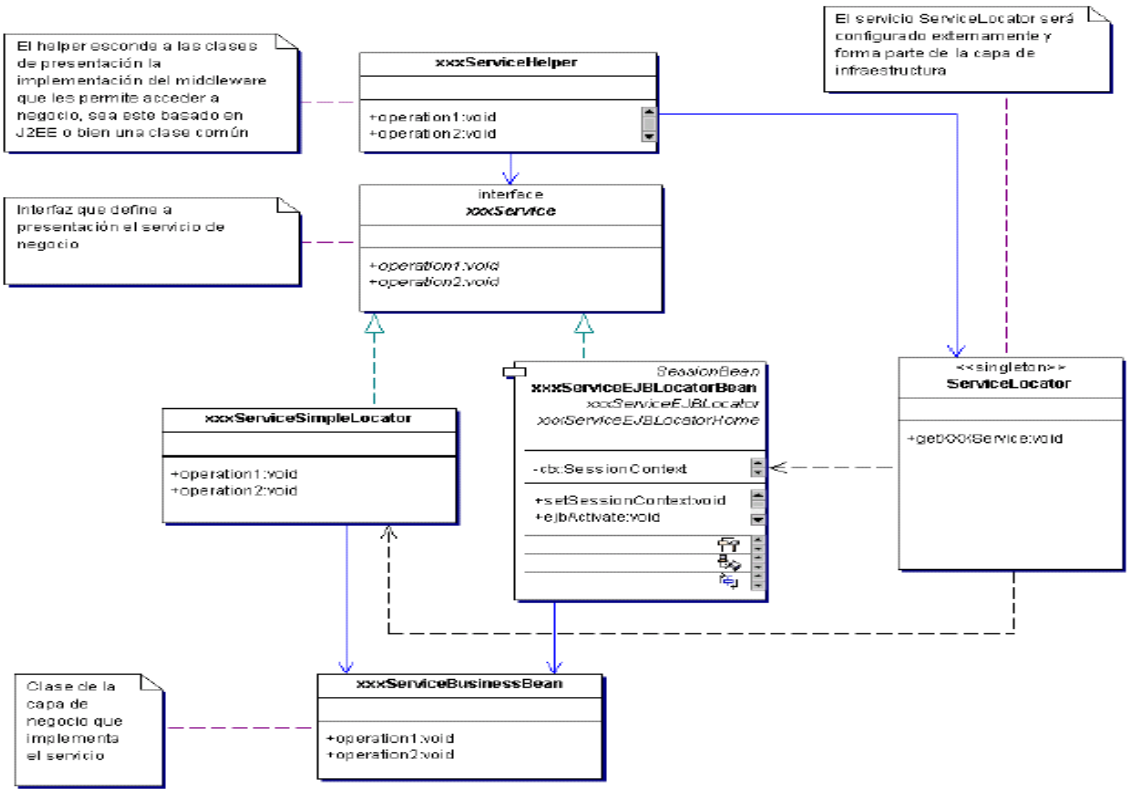
No obstante, este tipo de arquitectura presenta dos problema importantes cuando la aplicación no está distribuida verticalmente y se encuentra desplegada en una sola máquina:

1. De rendimiento, debido a que se están realizando invocaciones RMI o IIOP al localhost por cada solicitud de servicio que se realice de una capa a otra. En respuesta a esta problemática, *Sun Microsystems* decidió extender la especificación de EJBs con la incorporación de los interfaces locales. De esta forma, el desarrollador conserva la separación de responsabilidades, dado que sigue trabajando con el mismo interfaz del servicio, pero con la salvedad de que lo que se está realizado por debajo no es una llamada remota, sino una simple invocación local de la clase que implementa el interfaz.

2. Dependiendo de un contenedor de EJBs, pese a que su aportación al sistema no es necesaria en absoluto, limitando así la portabilidad del producto e imponiendo al cliente la incorporación a su sistema de un contenedor de EJBs. Solucionar este problema nos llevaría a otro que es configurar la aplicación para desplegarse en una sola maquina o en varias.

Entonces se opta solución alternativa que, limitándose a tareas de configuración del producto, permita ser desplegado tanto en sistemas no escalables verticalmente (es decir, careciendo de la separación física de las capas por medio de EJBs) como en los que si cuentan con un contenedor de EJBs y permiten dicha escalabilidad.

El siguiente patrón *Business Delegate* ofrece una solución acorde con estas premisas, en el diagrama se muestra un ejemplo de cómo aplicar dicho patrón a la separación entre la capa de presentación y la capa de negocio. En él se muestran las relaciones existentes entre las clase necesarias para un escenario bien sin distribución vertical, o bien con ella empleando EJBs de tipo sesión sin estado a modo de fachadas.



La función de los distintos elementos del diagrama será:

- **xxxServiceHelper**

El service helper funciona como elemento encapsulador del puente entre el servlet de presentación y el business bean en negocio. Simplifica la resolución del medio para llegar a la clase de negocio. Por medio del ServiceLocator (servicio de la capa de infraestructura), recupera la clase que implementa la interfaz del servicio de negocio, en base a la configuración actual de la capa de infraestructura. Esta puede ser bien una clase normal que simplemente invoque lo métodos del bean de negocio (modelo sin distribuir), bien un Enterprise JavaBean de tipo Session (modelo distribuido), o incluso un cliente RMI, cliente SOAP, etc. En el caso de un modelo sin distribución vertical, se tratará de una clase normal, permitiéndose así el despliegue del producto en una arquitectura física desprovista de contenedores de EJBs.

- **ServiceLocator**

A esta clase se le pide que resuelva cual es el componente que, en base a un fichero XML de configuración de la capa de infraestructura, debe instanciar en un momento para la interfaz que se le solicita. Será esta la que le sirva al helper el bridge a la clase de negocio. Habitualmente, se tratará de un singleton. Al estar configurada por medio de un fichero externo, el administrador del sistema puede establecer el bridge a usar (es decir, la clase que debe implementar la interfaz del servicio en cada ejecución) editando un fichero xml plano, y permitiendo así adaptar el sistema a distintos entornos sin necesidad de recodificar o repaquetizar el producto.

- **xxxServiceLocator**

Cuando el puente sea una clase simple, el ServiceLocator retornará una instancia de esta clase, que se limita a llamar a los métodos del bean de negocio cuando se invoquen los correspondientes a través del interfaz. En el diagrama esta clase aparece como *xxxSimpleService*Locator. En caso de una distribución basada en EJBs, se retornaría el stub de cliente, el cual invocaría a bean de implementación del EJB, que a su vez invocaría el bean de negocio. De esta forma, es transparente al implementador de la capa de presentación la forma con la que se invoca la capa de negocio.

1.5.4 Persistencia de entidades con EJBs entidad.

Los EJBs de entidad están pensados para ofrecer una visión orientada a objetos de una base de datos relacional. Una vez que se localiza la entidad en el repositorio por medio de los métodos **find** del EJB, el objeto distribuido mapea una tupla de la tabla que representa, siendo las actualizaciones que se realicen sobre sus atributos reflejados en la base de datos en el momento en el que el contenedor de EJBs lo considere necesario.

Esta visión del mapeo objeto relacional es académicamente purista, y libera al programador del sistema de la labor de desarrollar sentencias SQL, así como lo independiza de la base de datos que se encuentre tras el EJB, dado que es posible desplegarlo contra distintos repositorios de información sin necesidad de tocar ni una línea del código de las clases que lo utilizan. Sin embargo, presenta serios problemas de rendimiento, dado que los métodos de búsqueda aportados hasta la versión 1.2 de la especificación no contemplaban la posibilidad de realizar búsquedas cruzadas entre entidades (JOINS en SQL), lo cual obligaba a traspasar operaciones habitualmente resueltas eficientemente por el motor de

base de datos al código, realizando tareas tediosas de programar y con muy bajo rendimiento debido al excesivo uso de memoria.

Así mismo, la posibilidad de una solución híbrida entre el empleo de EJBs de entidad para el manejo de entidades individuales, y sentencias SQL para el procesado masivo de información del repositorio es inviable, dado que para que un EJB de entidad funcione adecuadamente, ha de garantizarse que todos los accesos a la tabla que mapea se realizan a través del propio EJB. Esto se debe al retardo en la sincronización con el repositorio que se aprovecha para aumentar el rendimiento del contenedor a modo de caché.

En la versión 2.0 de la especificación se ha tratado de poner solución a este inconveniente, permitiendo el establecimiento de relaciones maestro-esclavo entre EJBs, extendiendo el lenguaje EJB-QL e incorporando a la edición J2EE de la api JDO.

No obstante, aún no es posible alcanzar la potencia de ejecución que aporta el empleo del lenguaje SQL nativo de la base de datos objeto. La solución para la capa de persistencia más aceptada hoy en día es aquella que combina **EJBs de tipo sesión sin estado** a modo de fachadas carentes de lógica, con los DBBeans que atacan con sentencias SQL a la base de datos relacional sobre la que habitualmente se soporta el sistema de información.

1.6 Marco de Trabajo

1.6.1 Introducción

Al igual que se hizo anteriormente con los patrones, se introduce ahora el concepto de Framework para continuar la explicación de solución adoptada.

1.6.2 Definición

Un *framework* es un conjunto de herramientas para hacer frente a un problema en un contexto determinado.

Como herramientas se entiende: recomendaciones de diseño, software especializado, librerías de funciones... Tradicionalmente en programación orienta a objetos un framework esta formado por la extensión de un lenguaje mediante una o más jerarquías de clases que implementan una funcionalidad y que (opcionalmente) pueden ser extendidas.

El concepto de framework a tomado especial fuerza con el auge del software libre, que a menudo implementa frameworks de uso libre para problemas comunes y poder implementar soluciones empresariales.

1.7 Model View Controller

1.7.1 Introducción

En este apartado se presenta el Modelo Vista Controlador. Aunque el concepto MVC esta bien definido, qué es *MVC* es algo confuso, para algunos es un *patrón de diseño* y para otros un *framework*. En este documento se entenderá que es: *un patrón que como solución al problema que describe ofrece un diseño y nociones de cómo implementarlo*, *diseño tan extendido que ha llegado convertirse como un marco de trabajo para cualquier desarrollador de aplicaciones distribuidas*. **En cualquier caso no hay confundirlo con la arquitectura (física o lógica) de tres capas**.

1.7.2 Objetivo

El objetivo del aparatado es introducir las claves del diseño que se ha seleccionado para llevar a cabo el proyecto.

Se desarrolla una breve reseña de las características de este modelo, para dar paso a la descripción de la implementación.

1.7.3 Definición

Contexto: Aplicaciones distribuidas. Problema: Las aplicaciones son particularmente difíciles de portar a otros sistemas, probar y mantener. Solución: Hacer una distinción clara de responsabilidades e implementarlas por separado.

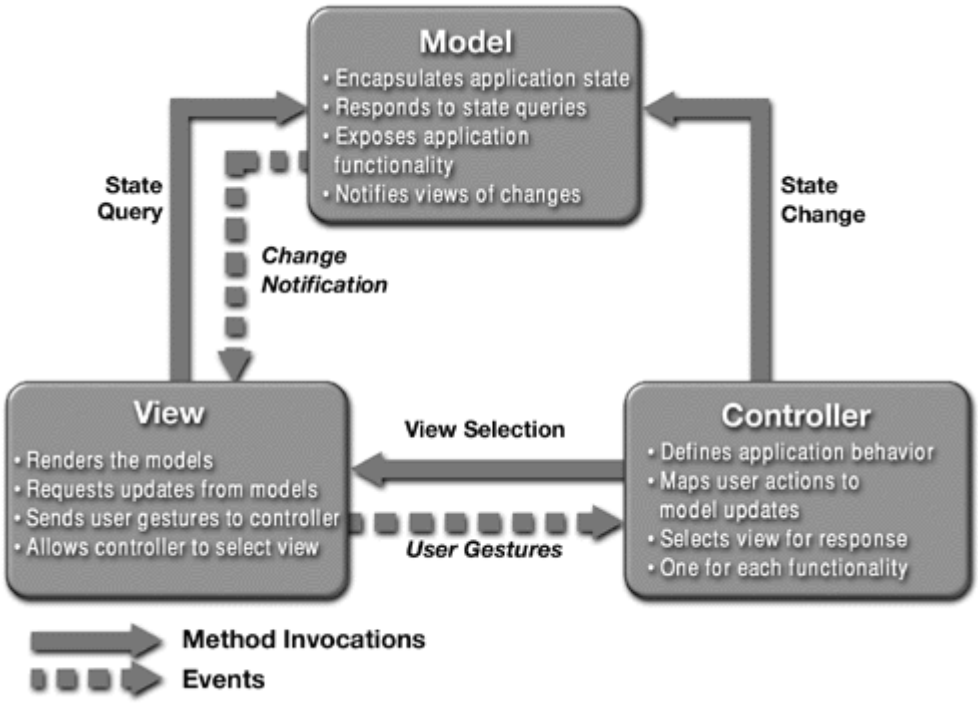
1.7.4 Descripción general

La arquitectura del MVC tiene sus raíces en Smalltalk, quien originalmente lo utilizó para realizar la conexión entre las tareas tradicionales de *entrada*, *proceso* y *salida*, con el modelo gráfico del usuario. Sin embargo, su aplicación al dominio de las aplicaciones web es intuitivo:

Este diseño consta de tres conceptos bien diferenciados como se muestra en la siguiente figura, de hecho la clave de este diseño consiste en diferenciar bien cada uno de los conceptos, lo cual no es tan fácil de implementar.

Se ha seleccionado el patrón MVC como guía para diseñar una solución escalable, estable y eficiente. El MVC ofrece los siguiente beneficios:

- Separar la vista del modelo permite que existir varias vistas sobre el mismo modelo, por lo tanto, es fácil y barato construir interface para distintas plataformas cliente.
- Mantener aislado el modelo permite facilitar la depuración y prueba de la lógica de negocio.
- Hace posible una auténtica separación de roles de desarrollo (diseñadores, programadores, funcionales, expertos en negocio...)



- **Modelo (Model)**

Representa la **información** de la aplicación conjuntamente con las **reglas** del negocio, las cuales permiten el acceso y modificación de la misma.

La lógica que implemente el modelo no debe tener conocimiento real de sus vistas, es decir, saber como mostrarse; simplemente el sistema mantiene enlaces entre el modelo y las vistas y notifica a éstas últimas el modelo cambia su estado.

- **Vista (View)**

Presenta la información que mantiene el modelo y es quien especifica la forma en que serán representados visualmente los datos.

Esta tarea puede ser llevada a cabo de dos formas:

- **push** en donde la vista se encarga de registrarse en el modelo para ser notificado de los cambios del mismo
- **pull** en donde la vista es responsable de invocar al modelo cuando requiere de información.

El usuario sólo debe interactuar con la vista.

- **Controlador (Controller)**

Es el encargado de traducir las interacciones con la vista, en acciones a ser ejecutadas por el modelo. Las acciones son el mecanismo para modificar el estado del modelo.

Como resultado de las acciones del usuario y de la posterior modificación del modelo, el controlador responde seleccionado y presentando la vista adecuada.

1.7.5 Evolución del MVC

La evolución tecnológica que el sector ha sufrido durante los últimos años ha permitido otra evolución paralela, la de la arquitectura de las aplicaciones web. A medida que aparecían nuevos recursos técnicos, los patrones de diseño se amoldaban para aprovechar las nuevas característica que estás novedades ofrecían. De esta forma, el modelo arquitectónico de las aplicaciones de Internet ha sufrido dos grandes saltos desde la aparición de los primeros portales.

Los distintos modelos de aplicación sobre los que ha desarrollado el MVC son los siguientes:

- **Modelo 1**

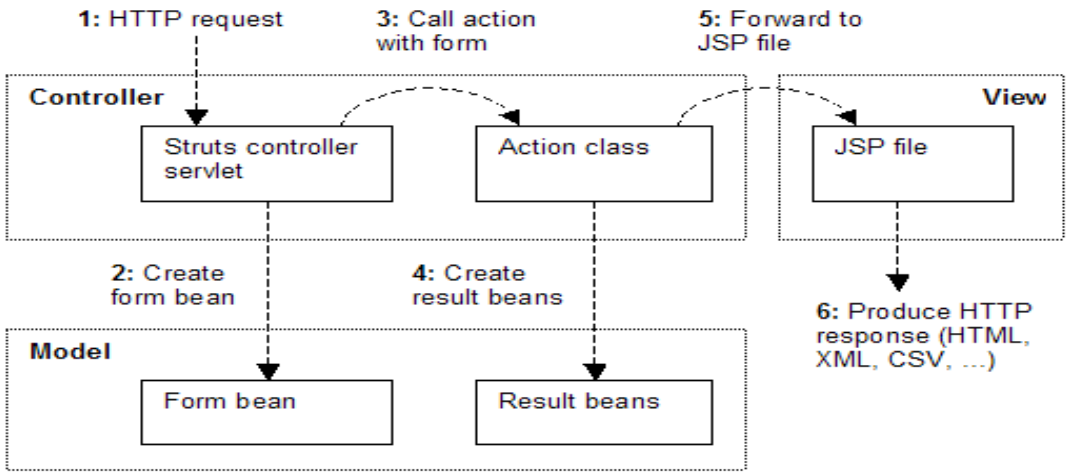
Son las más primitivas. Se identifican con este modelo las clásicas aplicaciones web CGI, basadas en la ejecución de procesos externos al servidor web, cuya salida por pantalla era el html que el navegador recibía en respuesta a su petición. Presentación, negocio y acceso a datos se confundían en un mismo script perl.

  - **Modelo 1.5**

Aplicado a la tecnología java, se da con la aparición de las páginas ASP de Microsoft, y posteriormente JSP y los servlets. En este modelo, las responsabilidades de presentación (navegabilidad, visualización, etc) recaen en las páginas dinámicas generadas en el servidor, mientras que los componentes incrustados en las mismas (javabeans, ActiveX, etc) son los responsables del modelo de negocio y acceso a datos.

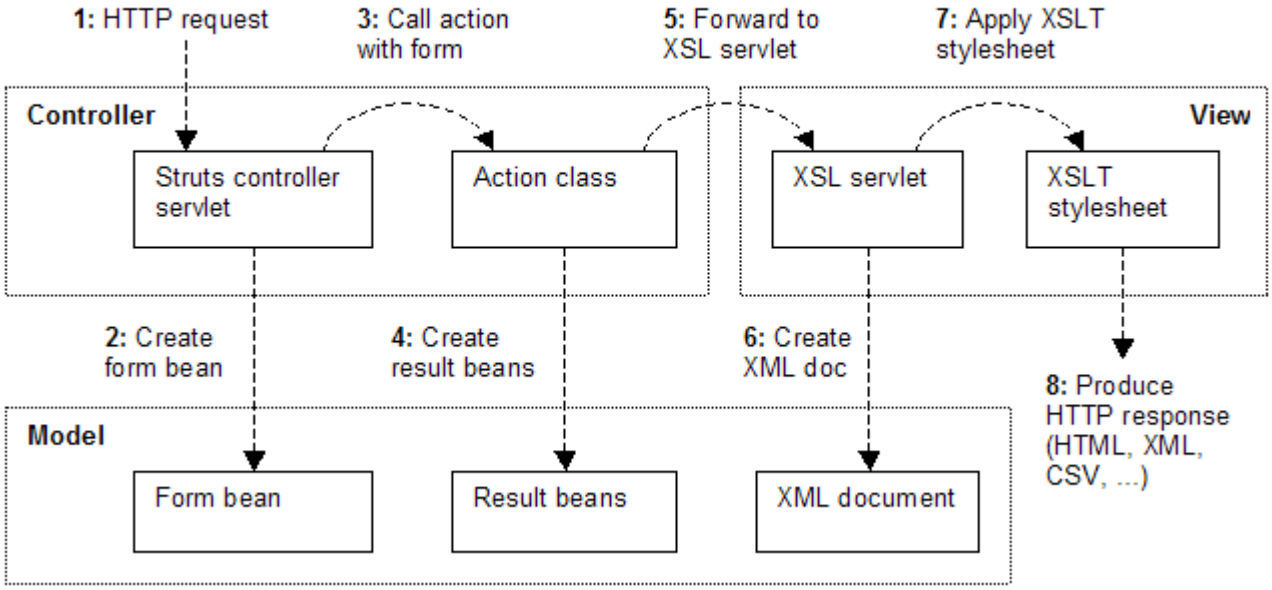
- **Modelo 2**

Como evolución del modelo 1.5 y con la incorporación del patrón MVC a este tipo de aplicaciones, se define lo que se conoce como *Model 2* de la arquitectura web. En el diagrama siguiente se aprecia la incorporación de un elemento controlador de la navegación de la aplicación. El modelo de negocio queda encapsulado en los componentes que se incrustan en las páginas de servidor.



o Modelo 2X

Con la implantación de XML, el modelo 2X aparece como evolución del modelo 2, con objeto de dar respuesta a la necesidad, cada vez más habitual, de desarrollar aplicaciones multicanal, es decir, con distintos tipos de clientes remotos (PC browser, móvil, PDA...).



Así, una aplicación web multicanal podrá ejecutarse desde una PDA, desde un terminal de telefonía móvil, o desde cualquier navegador html estándar. El medio para lograr publicar la misma aplicación para distintos dispositivos es separar de la Vista los datos del formato, por ejemplo empleando plantillas XSL estáticas, que representa el formato, para transformar los documentos XML dinámicos que son los datos y determinar la plantilla a emplear en base al parámetro user-agent recibido en la request. La aplicación de esta solución al modelo 2 de aplicaciones web define lo que se conoce como modelo 2X.

1.8 SOLUCIÓN PROPUESTA

1.8.1 Introducción

En este apartado se describe de un modo general, cual ha sido la arquitectura y diseño de la solución adoptada para llevar a cabo el proyecto. Una descripción más técnica y especifica será objeto de otro documento.

1.8.2 Arquitectura

Por los motivos indicados anteriormente se ha elegido una arquitectura de tres capas:

1. Presentación: La forma de visualizar la información será mediante páginas HTML internacionalizadas, visualizadas por un navegador web residente en la maquina cliente. La forma de entrar los datos será mediante formularios HTML que desencadene un petición http vía post.
2. Negocio: La lógica de negocio se implementará en JAVA siguiendo las especificaciones J2EE y residirá en el contenedor J2EE de la maquina servidor, aunque en principio el servidor es único y no esta previsto un escalamiento vertical, se llevará a cabo una implementación que permita incluir fácilmente objetos distribuidos.
3. Acceso a datos: Los datos residirán en el sistema gestor de base de datos Oracle (8i/9i) y acceso a ellos se llevará a cabo mediante un DataSource vía JDBC. Oracle DB residirá en la misma maquina u otra que el servidor J2EE.

1.8.3 Diseño

Se ha adoptado el diseño propuesto por el patrón MVC, en concreto el modelo 2X para facilitar la portabilidad multicanal. Para la implementación de este diseño se utilizarán otros patrones propuestos en el catalogó de patrones J2EE.

1.8.4 Implementación

Para llevar a cabo la implementación se ha elegido **Struts** uno de los framework más recomendados por la comunidad OpenSource para el desarrollo de aplicaciones web, en concreto la extensión **STxx** que proporciona soporte para el modelo 2X.

1.9 STRUTS y STXX

1.9.1 Introducción

A continuación se hace una descripción general del framework Struts, introduciendo la extensión utilizada Stxx, el objetivo es introducir la forma en la que este framework implementa el MVC-2X.

Las figuras del apartado "Evolución del MVC" describen perfectamente el diseño de Struts y la extensión Stxx.

1.9.2 Funcionamiento

El funcionamiento clásico de *Struts* es el siguiente:

1.- El usuario hace una petición http.

2.- El controlador en base al fichero de configuración y la petición recibida toma una decisión y ejecuta una serie de acciones.

3.- Las acciones utilizan la lógica de negocio (EJB...) para modificar el estado de la sesión del usuario, los datos de la aplicación y por último generar un resultado.

4.- El controlador en función del resultado de la acción (éxito, fracaso...) y del fichero de configuración decide que vista debe invocar (normalmente un JSP).

5.- El JSP invocado utiliza muestra el estado de la sesión (modificado por las acciones) al usuario.

6.- El usuario interactúa con la vista y volvemos al punto 1.

El funcionamiento de la extensión *Stxx* es común en los dos primeros puntos:

3.- Las acciones además de generar un resultado, guardan en la sesión los datos modificados en formato XML.

4.- El controlador en función del resultado de la acción (éxito, fracaso...) y del fichero de configuración decide que vista debe generar, para ello envía al motor de transformación una plantilla XSL y los datos XML generados o modificados por las acciones.

5.- El motor XSLT, transforma en función del fichero de configuración los datos XML en otro XML con formato, normalmente XHTML y se lo envía al navegador del usuario

6.- El usuario interactúa con la vista y volvemos al punto 1.

1.9.3 Controlador

Struts/stxx se basa en el patrón FrontController para implementar el controlador. El núcleo del "Controller" esta constituido por un servlet que lee la cofiguración de un fichero XML, en cual se describen que acciones corresponden a cada petición de un usuario y que vistas corresponden a cada acción.

1.9.4 Modelo

Struts no implementa el "Model" ya que esta capa es particular de cada aplicación pero proporciona soporte facilitar la integración de la lógica de negocio en el framework, como son los ActionForm y otros bean.

1.9.5 Vista

La vista tradicional de Struts esta implementada por JSPs y Taglibs, sin embargo, la extensión Stxx la reemplaza por un motor de transformación XSL.

1.9.6 Características

Esntre la características principales de Struts destacan:

- Proporciona soporte para la internacionalización de la vista por medio de mensajes formateados en ficheros de propiedades y objetos Locale de Java.
- Proporciona soporte para la gestión de errores mediante taglibs y una jerarquia de clases.
- Proporciona una forma fácil de logar de información mediante Log4J
- Proporciona un fácil acceso a base de datos, mediante la configuración de DataSources.
- Además esta respaldado por una comunidad enorme de desarrolladores que continuamente aportar nuevas funcionalidades, como la integración de JAAS en Struts...

2. TECNOLOGÍAS UTILIZADA

2.1 Introducción

Esta sección define todos las tecnologías y sus requisitos que forman parte del proyecto con el objetivo de facilitar su implantación y mantenimiento.

- **Servlet y JSP**

Tanto las páginas JSP como los servlets producen contenido dinámico por medio de la ejecución de código java en el servidor de aplicaciones cada vez que son invocados.

La diferencia entre ambos es que JSP está escrito con una versión extendida de HTML, y los servlets son escritos en Java.

JSP es conveniente para los diseñadores Web, quienes conocen HTML y están acostumbrados al trabajo en editores HTML. Por otro lado, los servlets, escritos enteramente en Java son más convenientes para programadores Java. Escribir un servlet requiere de algún conocimiento del protocolo HTTP y de programación Java.

- **TagLibs**

Las librerías de tags permiten encapsular acciones comunes a varias páginas en un módulo reutilizable. Esto permite que las JSP que componen una aplicación compartan una forma standard de resolver esas acciones, proporcionando al diseñador WEB una forma cómoda de invocar código java y facilitando al programador java la codificación de acciones.

- **EJB -Enterprise JavaBeans-**

La especificación de JavaBeans Enterprise define una arquitectura para el desarrollo y despliegue de aplicaciones basadas en objetos distribuidos transaccionales, software de componentes del lado del servidor. Las organizaciones pueden construir sus propios componentes o comprarlos a vendedores de terceras partes. Estos componentes del lado del servidor, llamados beans enterprise, son objetos distribuidos que están localizados en contenedores de JavaBean Enterprise y proporcionan servicios remotos para clientes distribuidos a lo largo de la red. Los interfaces remoto y home son tipos de interface "Java RMI Remote" . El interface java.rmi.Remote se usa con objetos distribuidos para representar el bean en un espacio de direccionamiento diferente (proceso o máquina). Un bean enterprise es un objeto distribuido, esto significa que la clase bean es ejemplarizada y vive en un contenedor pero puede ser accedida por aplicaciones que viven en otros espacios de direccionamiento.

Internamente, los EJBs pueden emplear tanto RMI como IIOP (protocolo definido por CORBA), aunque esto es totalmente transparente al usuario del objeto remoto, dado que ni siquiera se percatará de si el objeto se encuentra en su misma máquina o en otra a miles de kilómetros de distancia.

o EJBs de tipo Entidad

El bean de entidad es uno de los tres tipos de beans primarios. El bean de entidad es usado para representar datos en un base de datos. Proporciona un interface orientado a objeto a los datos que normalmente serían accedidos mediante el JDBC u otro API.

Entity beans pueden participar en transacciones que involucren otros enterprise beans y servicios de transacción. Estos beans son a menudo mapeados a objetos de la base de datos. Un entity bean puede representar una fila en una tabla, una columna en una fila, o una tabla entera o el resultado de un query. Asociado a cada entity bean existe una clave primaria utilizada para buscar, sacar y salvar el bean.

Un Entity bean puede tener persistencia Bean-managed, por la cual el bean contiene código para tomar y salvar los datos persistentes. O bien persistencia Conteiner-managed, por la cual el contenedor EJB es quien carga y salva los datos a nombre del bean; cuando este tipo de persistencia es utilizada, el compilador EJB genera el soporte JDBC necesario para el mapeo.

Los entity beans pueden ser compartidos por muchos clientes y aplicaciones. Una instancia de un entiry bean pued ser creada por el request de un cliente, pero no desaparece cuando el cliente se desconecta. Continúa su vida mientras un cliente lo esté usando. Cundo ya no es más usado, el contenedor de EJB dispone de él.

o EJBs de tipo Sesión

Los EJBs de tipo sesión son objetos distribuidos puros, cuya implementación no ha de estar ligada a ningún sistema de persistencia de entidades, como en el caso anterior. Responden exactamente al modelo de objetos de CORBA, con la salvedad de que en su creación fuerzan la aplicación del modelo factoría[3]. Desde el punto de vista del usuario del bean, contará con una clase cuyos métodos podrá invocar de forma habitual, sin conocer si la ejecución del método se realiza en la misma máquina o en otra distinta. Dentro de los EJBS de sesión se distinguen dos tipos.

- Con estado – **Statefull** -

El objeto remoto es capaz de garantizar que su estado va a mantenerse para el mismo usuario del objeto entre distintas invocaciones. Así, si en la invocación 1 se cambia el valor de uno de los atributos del objeto, en la siguiente invocación desde la misma clase, el valor nuevo del atributo se conservará. El mantenimiento de este estado implica la serialización del estado del bean cuando el objeto remoto es compartido (en un pool, habitualmente) por varios clientes remotos, lo cual supone un coste importante que pone en duda el posible rendimiento de este tipo de solución.

- Sin estado – **Stateless** -

Los EJBs de sesión sin estado no garantizan el mantenimiento del estado del objeto remoto entre dos invocaciones sucesivas del mismo cliente. Esto sujeta al diseñador del sistema a mantener el estado de la sesión en la parte del sistema que esté por encima de la capa delimitada por los objetos distribuidos.

No obstante, pese a este inconveniente, se facilita la reutilización de objetos en pools, y se evitan los posibles problemas de rendimiento derivados de la serialización contínua del estado de los beans.

- **XML**
- **XSL**
- **JAXP / JAXB**
- **JDBC**
- **JAAS**
- **Struts/Stxx**
- **Log 4J**
- **JDK 1.4**

Bibliografía

[GOF94]
Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (The Gang of Four). *Design Patterns*. Addison Wesley Professional Computing Series, 1994.
[PATTSUN]
http://java.sun.com/blueprints/corej2eepatterns/index.html
[PATTPC]
http://www.programacion.net/tutorial/patrones/

[JAKA03]
http://jakarta.apache.org/
[STRUTS]
http://jakarta.apache.org/struts/
[STXX1]
http://stxx.sourceforge.net/
[STXX2]
http://www.orbeon.com/oxf/doc/model2x-model2x
[STPC1]
http://www.programacion.com/tutorial/struts/
[STPC2]
http://www.programacion.net/articulo/tips_struts/
[MERC02]
Julien Mercay and Gilbert Bouzeid.
Boost Struts With XSLT and XML. Java World, Febrero 2002.

[WEBSA]
http://www.osmosislatina.com/aplicaciones/servidor_web.htm

CONTROL DE VERSIONES

Este documento no esta cerrado y se irá ampliando según sea necesario hasta finalizar el proyecto.

1.0 → Arquitectura; J2EE; MVC
1.1 → Gráficos; Patrones; Modelo 1, 2 y 2X
1.2 → Objetos distribuidos; STRUTS y STXX
1.3 → Apéndice 1 y Apéndice 2
1.4 → Arquitectura adoptada
1.5 → Tecnología
2.0 → Formato

APÉNDICE 1

1. Introducción

Este documento pretenden distinguir los conceptos:

- Servidor Web (Web Server) o servidor de páginas
- Servidor de Aplicaciones (Application Server)
- Contenedor Servlet (Java Engine)
- Contenedor EJB (J2EE Server)

2. Servidor Web

El servidor de Páginas es la parte primordial de cualquier sitio de Internet, ya que es el encargado de generar y enviar la información a los usuarios finales.

Cuando se crearon los primeros Servidores de páginas ("Web Server") como Apache, éste solo era encargado de enviar los datos al usuario final, pero cualquier otra información que requiriera de algún tipo de personalización era realizada por un interpretador que ejecutaba un "script" (programa), generalmente en Perl. Sin embargo, conforme las demandas de los Servidores de páginas incrementaron fue necesario mejorar este proceso, ya que el llamar un interpretador para que ejecutara otro programa ponía una demanda muy fuerte sobre la maquina que mantenía el Servidor de Páginas (el Host).

Típicamente un web server debe afrontar las peticiones http de un navegador (Internet Explorer, Netscape) y servirle la información que desea, normalmente páginas HTML.

Entre los Web Server más utilizados destacan Apache, AOL Server e IIE.

Apache

Es uno de los Servidores de páginas más utilizados, posiblemente porque ofrece instalaciones sencillas para sitios pequeños y si se requiere es posible expandirlo hasta el nivel de los mejores productos comerciales. Si se utiliza para un sitio pequeño que solo contenga archivos en HTML, esto es, no requiera de aplicaciones de servidor su funcionalidad es excelente.

Cuando el servidor de páginas (Apache) recibe la requisición para "x" página éste reconoce cuando debe enviar un documento estático (HTML) o ejecutar algún tipo de aplicación, en el diagrama se puede observar que la solicitud de "x" página invoca (llama) un programa en Perl y este a su vez solicita información a una base de datos, por lo tanto para llevar acabo esta operación debieron iniciarse *2 procesos* nuevos, esto implica una gran disminución de eficiencia al escalar las peticiones.

IIS (Information Server)

IIS es el servidor de páginas desarrollado por Microsoft para Windows NT/2000, solo puede operar en plataformas Windows. El punto más favorable de este servidor son ASPs que facilitan el desarrollo de aplicaciones y la "sencillez" de instalación, sin embargo, existen alternativas como ADP's de *Aolserver* y JSP's para Java.

3. Servidor de Aplicaciones

En un sentido *muy estricto* un "Web Server" no es lo mismo que "Application Server", pero últimamente estos dos términos se prestan a una gran confusión, ya que normalmente cualquier Servidor de Aplicaciones puede utilizarse como servidor web, aunque lo ideal es combinarlos.

Un servidor de aplicaciones esta especializado en servir contenido dinámico y un servidor web en servir contenido estático. En este sentido IIS podría ser considerado servidor de aplicaciones ya que da soporte a ASP...

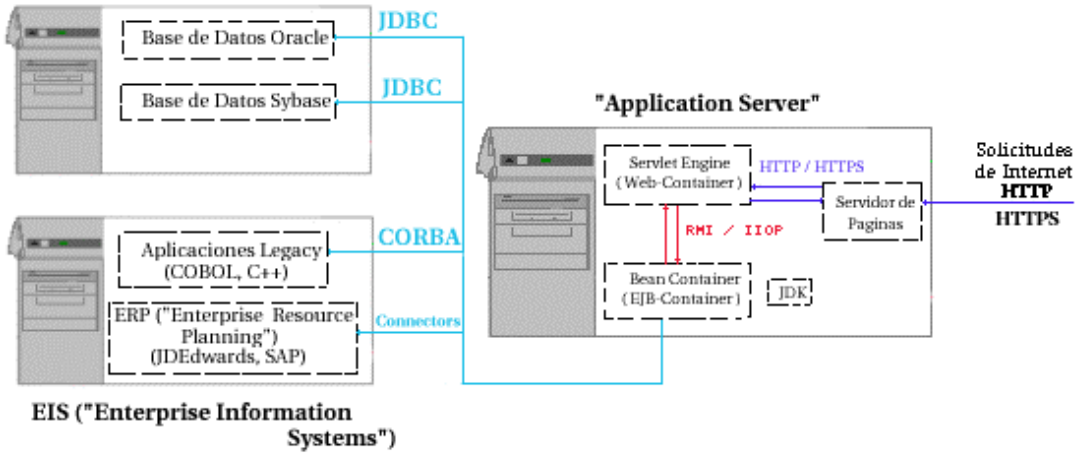
En la actualidad los servidores web no se utilizan por que todos son servidores de aplicaciones (Apache – Perl; AOL – ADP; IIS – ASP), sin embargo, conviene tener clara la diferencia a la hora de desarrollar aplicaciones web.

En la actualidad los servidores de aplicaciones no sólo sirven contenido dinámico sino que son responsables de mantener objetos en memoria, controlando su ciclo de vida, visibilidad... debido a esto han surgido dos conceptos nuevos: "Java Application Servers" que son otra cosa muy diferente y malamente designados como "Application Servers".

4. Java Application Server

"Java Application Servers" hoy en día ya denominados "Application Servers" ofrecen una manera de Integrar y ofrecer las funcionalidades requeridas por la gran mayoría de sistemas empresariales, una de las razones por las cuales el mercado ha sido inundado con estos "Application Servers" es que están diseñados alrededor de J2EE, que es un grupo de especificaciones definidas por Sun.

Estos Servidores de Aplicaciones Java comunmente llamados **Middleware** se encuentran compuestos de la siguiente manera:



Como su denominación lo implica (**"Middleware"**) se encuentran en la parte media de una arquitectura de sistema, su flexibilidad reside en la posibilidad de acceder a información en sistemas empresariales (EIS) como SAP, JdEdwards, bases de datos o alguna aplicación escrita en otro lenguaje.

Dependiendo de la empresa que desarrolle el "Application Server" éste puede contener inclusive hasta un "Servidor de Páginas" o algún otro desarrollo propietario, sin embargo, los dos elementos primordiales (aunque no sean comercializados como tal) son el **"Servlet Engine" (Web-Container) y "Enterprise Bean Engine" (Bean-Container)**.

5. Servlet Engines o Contenedor Web

Quizás el nombre que más salga a relucir con "Servlet Engines" es *Tomcat* o *Jakarta Apache*. Tomcat surgió de Sun Microsystem's cuando desarrollaban un "Servidor de Páginas" que utilizara "Java",y posteriormente cedieron el código fuente a la fundación Apache.

A pesar del nombre Apache-Tomcat; Tomcat no requiere de Apache para su funcionamiento (solo requiere de un jdk) y es aquí donde se diferencia se Apache:

Tomcat es capaz de responder a requisiciones de Internet, en efecto actuando como "Servidor de Páginas", sin embargo, aunque esto sea posible la gran mayoría de las implementaciones de Servlet Engines no funcionan tan eficiente como un "Servidor de Páginas", es por esto por lo que se recomienda utilizar un "Servidor de Páginas" (Apache, Aol, IIS) en conjunción con un "Servlet Engine" (Tomcat, Resin...) mediante el protocolo ajp12 o ajp13.

Que hace el Servlet Engine ?

El "Servlet Engine" ofrece un **"Ambiente"** donde habitan los JSP y Servlets, es ahí donde se contemplan una gran cantidad de funcionalidades como: threading, manutención de sesiones, conectividad con el "Servidor de Páginas", es por esto al "Servlet Engine" también se le denomina "Web-Container".

Dos "Servlet Engines" (Web-Containers) que están en amplio uso y son utilizados con "Servidores de Páginas" son: Tomcat y ServletExec, donde el primero es open-source y el último es un producto cerrado; otro "Servlet Engine" es Resin (Open-Source) el cual permite: utilizar JavaScript como "Scripting Language" dentro de JSP's y acceso a XSL.

6. Servidor J2EE, Contenedor EJB

El "Enterprise Bean Engine" (Bean-Container) ofrece un **"ambiente"** donde residen los EJBs, es mediante "Enterprise Java Beans" que se ejecuta la *lógica de negocios* sobre la información que reside en los sistemas empresariales ("EIS"). En el "Bean Container" (al igual que en el "Web Container") se contemplan varias funcionalidades: "Pooling" hacia bases de Datos (JDBC), control de transacciones (JTA-JTS), conectividad con ERP(Connectors), aplicaciones legacy (CORBA),entre otras cosas.

La mayor ventaja de este tipo de arquitectura se debe a la separación de funcionalidades y uso de protocolos de red como RMI/CORBA, esto facilita que puedan existir 4 o 5 "Hosts" en diferentes regiones geográficas, cada uno empleando cualquiera de los componentes antes mencionados.

Por último, existen diversos "Application Servers" que son denominados **"Fully J2EE Compliant"** esto indica que cumplen con todas las especificaciones J2EE indicadas por Sun. (Vea J2EE). Algunos "Application Servers" **"Fully J2EE Compliant"** son WebLogic de BEA, WebShere (WAS) de IBM, Jrun, OIAS de Oracle, iPlanet de Netscape o JBOSS (open-source).

APÉNDICE 2

1. Concepto de Pattern

La idea de *patrones de software* tuvo su origen del campo de la arquitectura. Christopher Alexander, un arquitecto, escribió dos libros revolucionarios que describían patrones en arquitectura de construcción y planificación urbana: *A Pattern Language: Towns, Buildings, Construction* (Oxford University Press, 1977) y *The Timeless Way of Building* (Oxford University Press, 1979).

Las ideas presentadas en estos libros son aplicables a varios campos además de la arquitectura, incluyendo el software. En 1987, Ward Cunningham y Kent Beck usaron algunas de las ideas de Alexander y desarrollaron cinco patrones para el diseño de interfaces de usuario (UI), que publicaron en un artículo de OOPSLA'87 llamado *Using Pattern Languages for Object-Oriented Programs*. En 1994 Erich Gamma, Richard Helm, John Vlissides y Ralph Johnson publicaron uno de los libros más influyentes de esta década: *Design Patterns*. Este libro, también llamado "Gang of Four" (o GoF), popularizó la idea de patrones en el diseño y construcción de software.

Posteriormente los ingenieros de Sun Microsystem crearon una serie de patrones para dar solución a los problemas más habituales que surgen al programar aplicaciones web. Estos patrones se conocen normalmente como Core Pattern J2EE, a continuación se describen los más utilizados, clasificados según el lugar donde se aloja la solución que aportan.

2. Patrones J2EE de la capa de presentación

2.1 Intercepting Filter

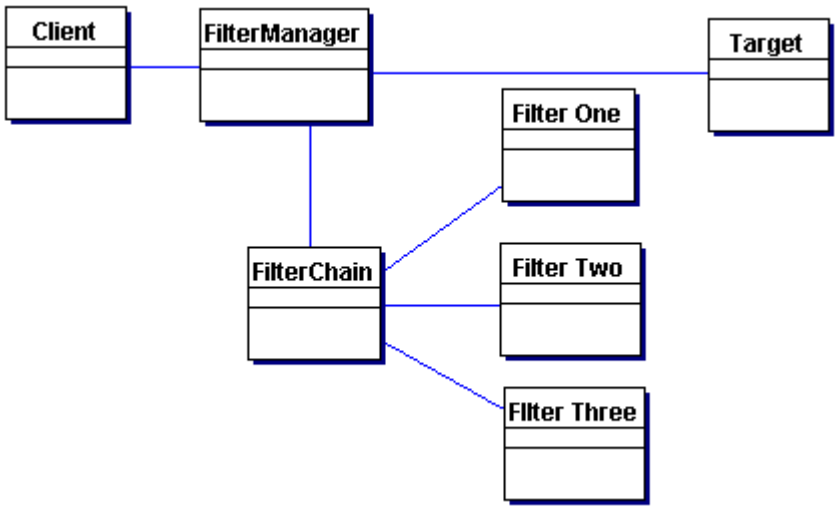
La capa de presentación requiere manejar mecanismos para recibir diferentes tipos de requerimientos, que pueden requerir variados tipos de procesamiento. Algunos de estos requerimientos son simplemente reenviados a la correspondiente componente que maneje este dato, mientras que otros requerimientos deben ser modificados, auditados o descomprimidos antes de continuar el procesamiento. Por ejemplo, cuando un usuario solicita una página Web, ésta puede requerir realizar varias validaciones previas antes de realizar el procesamiento principal de la página, ejemplo:

- El cliente ha sido autentificado?
- El cliente tiene una sesión válida dentro del sistema?
- La dirección IP del cliente, pertenece a una red confiable?
- El path requerido no cumple con alguna restricción?
- El sistema soporta la versión del browser del usuario?

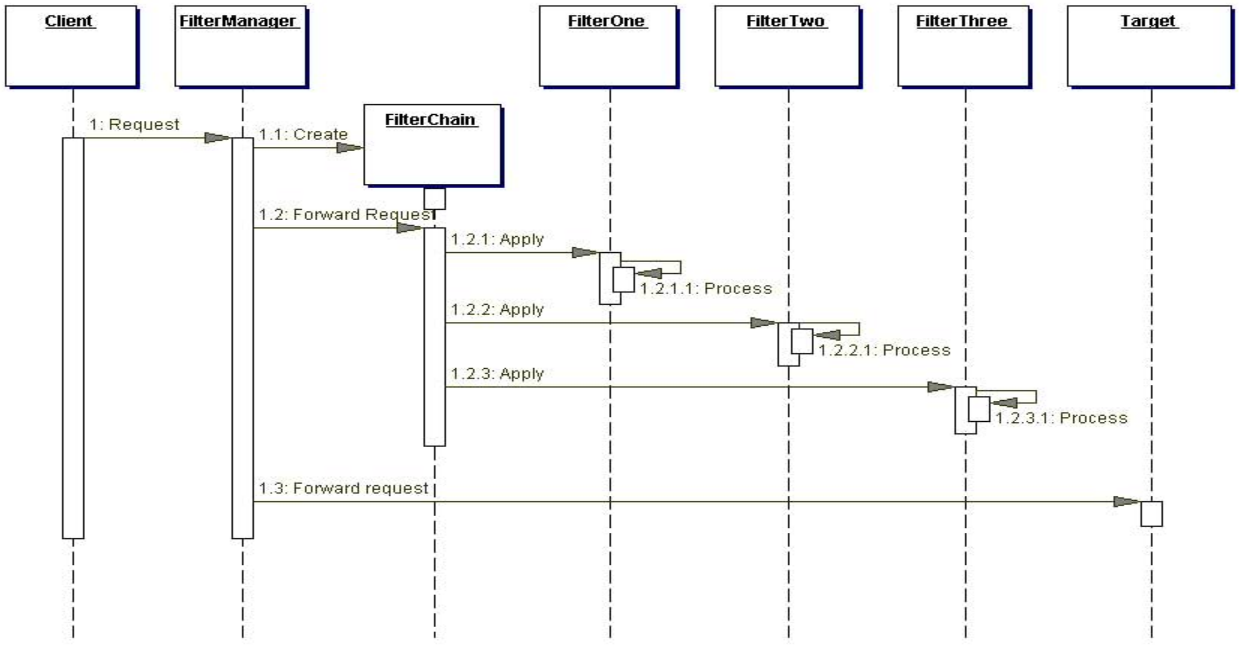
Mientras que algunos de estos chequeos son test del tipo SI o NO, otros requieren una modificación de la información de entrada a la página. La solución clásica a este problema es el uso de variadas condiciones del tipo if/else. Sin embargo, una solución de este tipo nos lleva a un código frágil y con un estilo de programación basado en "copiar y pegar".

La solución planteada por este patrón es crear filtros "pluggables" que procesen los servicios comunes en una forma estandard, de tal forma que no requiera efectuar cambios en el núcleo del código que procese los requerimientos. Los filtros interceptan los datos de entrada y las respuestas de salida, permitiendo un pre-procesamiento y un post-procesamiento. Además permiten agregar o eliminar filtros, sin requerir una modificación del código existente.

La estructura de clases para este patrón es la siguiente:



El siguiente es el diagrama de secuencia del patrón:



FilterManager: Maneja el procesamiento de los filtros. Crea el *FilterChain* con los filtros apropiados, en el orden correcto, e inicia el procesamiento.

FilterChain: Es una colección ordanada de filtros independientes.

FilterOne, FilterTwo, FilterThree: Son los filtros individuales que están mapeados a un objetivo.

FilterChain coordina su procesamiento.

Target: Es el recurso requerido por el cliente.

3. Patrones J2EE de la capa de negocios

3.1 Business Delegator

La capa de presentación interactúa directamente con la capa de servicios de negocios. Esta interacción directa expone detalles de la implementación de la API del servicio de negocio. Como resultado, la capa de presentación es vulnerable a cambiar en su implementación ante cambios en los servicios de negocio:

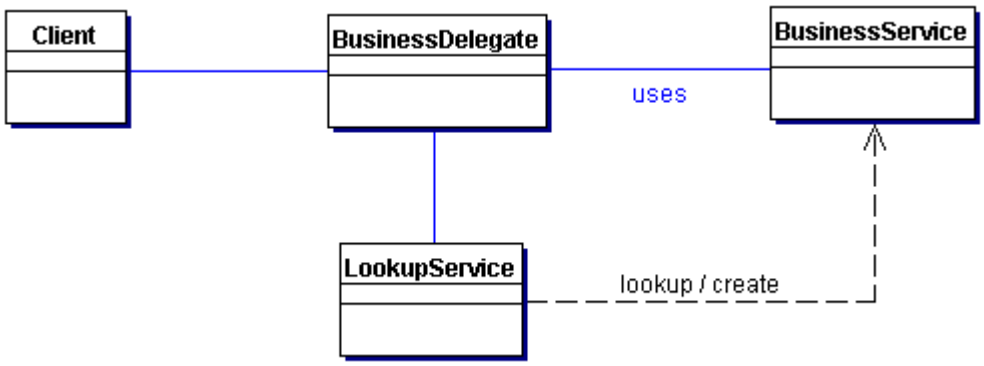
Cuando la implementación del servicio cambie, el código de la capa de presentación también cambiará.

Adicionalmente, esto puede provocar un impacto en el rendimiento de la red, puesto que las componentes de la capa de presentación que usen las API de los servicios de negocios realizan demasiadas invocaciones sobre la red. Esto sucede cuando los componentes de la capa de presentación usan la API directamente, sin mecanismos de cache por el lado del cliente o mecanismos de agregación.

Por último, exponer la API de los servicios directamente al cliente, fuerza al cliente a ocuparse de aspectos de la red asociados con la naturaleza distribuida de la tecnologia de los Enterprise JavaBeans (EJB).

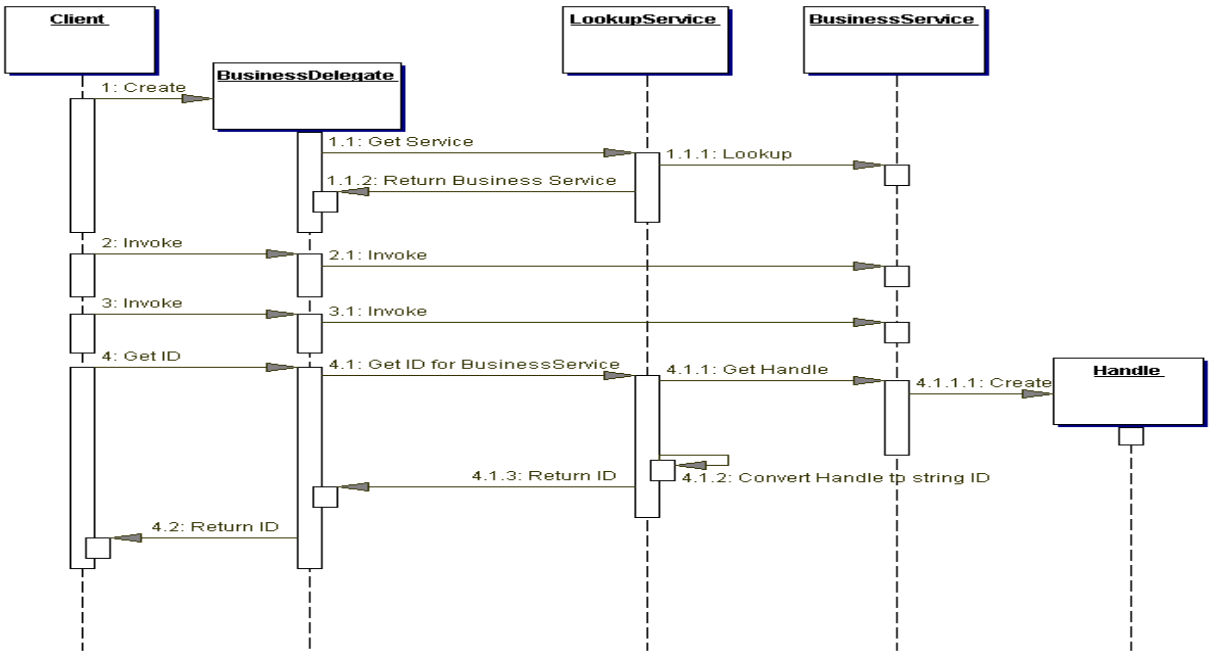
El patrón plantea usar un *Delegador de Negocio* (Bussiness Delegator) para reducir el acoplamiento entre los clientes de la capa de presentación y los servicios de negocios. El Delegador de Negocios esconde los detalles de la implementación del servicio de negocio, tales como opereciones de busqueda y detalles de acceso de la arquitectura EJB.

La siguiente figura muestra el diagrama de clases para este patrón:

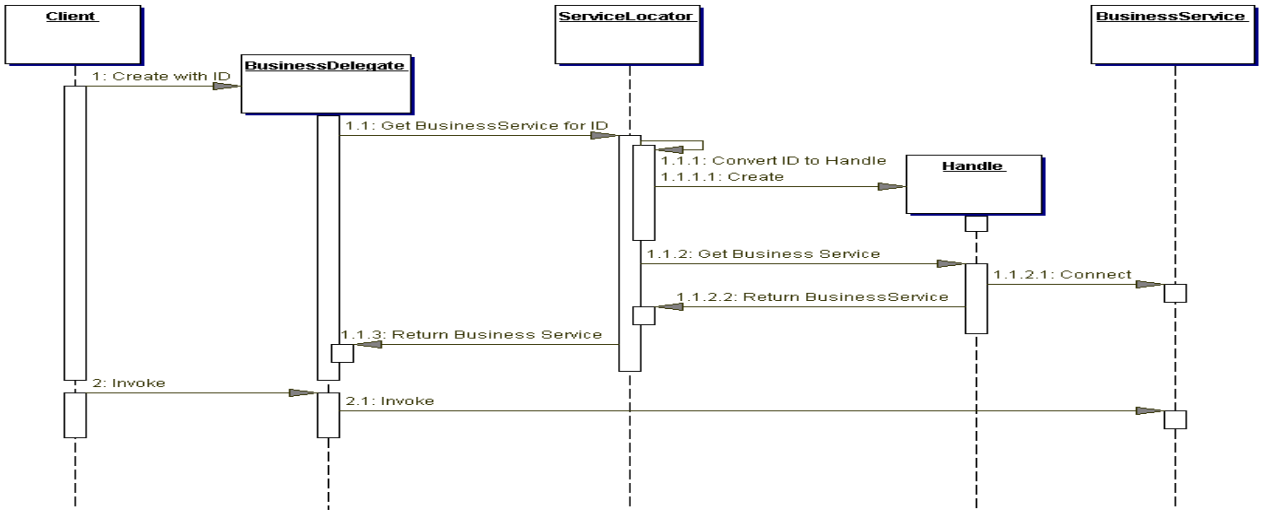


El cliente solicita a *BussinessDelegate* un requerimiento para proporcionar el acceso al servicio de negocio. El *BussinessDelegate* usa un *LookupService* para localizar el componente *BusinessService* requerido.

La siguiente figura muestra el diagrama de secuencia que ilustra la interacción típica de este patrón:



Una variación común de este patrón es usada con una estrategia de Adaptador. Esto permite que diversos sistemas puedan usar un XML como lenguaje integrador de todos los sistemas. La siguiente figura ilustra esta situación:



De esta forma, el cliente B2BClient envía sus requerimientos a B2BAdapter en un lenguaje genérico, como lo es XML, independiente del tipo de servicio que se desee consultar. B2BAdapter procesa el requerimiento y solicita la invocación del Servicio requerido a BussinessDelegate, quien es el que finalemente realiza la invocación solicitada.

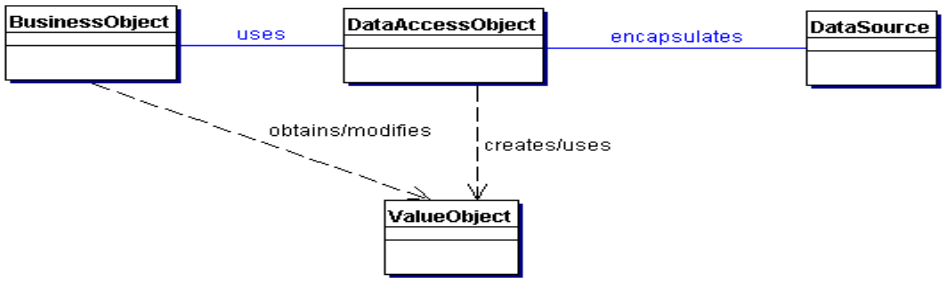
4. Patrones J2EE de la capa de integración

4.1 Data Access Object

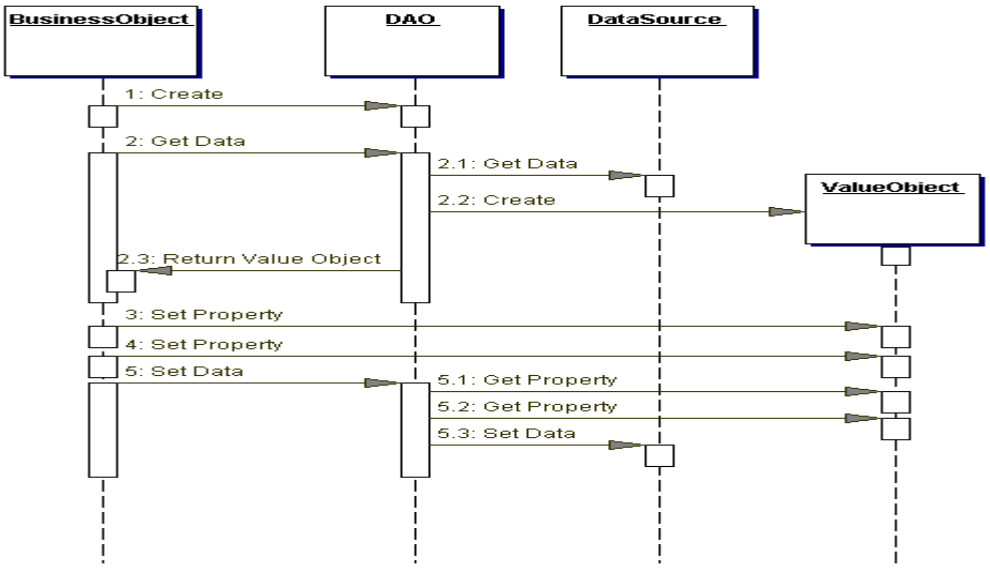
Muchas apicaciones requieren manejar persistencia de datos consultados a un determinado servicio. Sin embargo, los mecanismos de almacenamiento persistente con frecuencia son diferentes y dependen con frecuencia del tipo de servicio específico que se consulta. Por ejemplo, considere un sistema que requiera obtener información de fuentes de datos diferentes, como una Base de Datos Relacional, una Base de Datos Orientada a Objetos, un servicio LDAP, o que requiera obtener información de otros sistemas (B2B), para validar tarjetas de credito, etc.

Este patrón propone la utilización de un Objeto de Acceso a Datos (Data Access Data - DAO) para abstraer y encapsular todos los accesos a fuentes de datos. El DAO maneja y controla las conexiones con las fuentes de datos para obtener y guardar la información solicitada.

El siguiente es el diagrama de clases del patrón:



El siguiente es el diagrama de secuencia del patrón:



BusinessObject representa el objeto que requiere acceso a la fuente de datos para obtener cierta información almacenada. La solicitud es enviada a *DataAccessObject*, (el cual es el objeto principal del patrón), quien provee de un acceso transparente a la fuente de datos, en este caso *DataSource*. Una vez obtenida la información, ésta es almacenada en *ValueObject* para su utilización por parte de *BusinessObject*.

CAPA CONTROLADOR

Intranet para un Dpto. Universitario

Conceptos y Desarrollo de la Capa de Controlador

1 de Julio de 2004, v 2.2

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

1. PRÓLOGO

Este documento esta dirigido a todos aquellos que quieran comprender como funciona el flujo entre las distintas páginas de la Intranet.

El contenido del documento es comprensible para todos aquellos que tengan nociones básicas sobre aplicaciones web, contenedores de aplicaciones**[WEBSA].** Se recomienda leer el informe tecnológico a modo de introducción a este documento que es más técnico.

Como complemento a este documento se recomienda leer Gestión de Errores.

2. INTRODUCCIÓN

La capa del controlador **[PATTSUN]** es la base fundamental del núcleo de la aplicación y por ello la más desarrollada. De hecho, no es necesario o no debiera ser necesario modificar absolutamente nada de la capa del controlador, salvo los descriptores de flujo, para añadir nuevas páginas a la aplicación.

Uno de los objetivos del documento es informar de cómo añadir nuevas acciones al sistema, para ello bastará con una lectura rápida y la otra es explicar como funciona el framework en el que esta basada la implementación del controlador, para adentrarse en como funciona internamente el controlador, nos remitimos a los comentarios del código fuente.

La Intranet como cualquier otra aplicación WEB necesita de un sistema que conduzca al usuario de una página a otra y gestione los posibles errores que puedan ocurrir, este sistema es el controlador y para implementarlo se ha optado por un framework muy extendido en la comunidad de Internet: *Struts* **[STRUTS]** y su extensión *StrutsCX*. **[STCX1]**

3. Lógica del Controlador

Como ya se ha mencionado el controlador es un conjunto de clases y recursos que se encarga de ejecutar la lógica de negocio necesaria y ofrecer los resultados a la vista de acuerdo a las peticiones del usuario.

El controlador esta basado en un servlet configurable mediante descriptores XML y conjunto de acciones. De hecho es el servlet el verdadero controlador y las acciones son el nexo entre la capa del controlador, el modelo y la vista.

El funcionamiento es muy simple, cada petición del usuario esta asociada a una acción que hace las llamadas necesarias al modelo para generar un XML que es traspasado a la capa vista para que este sea presentado al usuario.

Una acción puede desencadenar una o varias llamadas al modelo o ninguna y simplemente cargar una página estática.

Una acción puede llamar a otras acciones que completen la petición del usuario, así pues una acción puede validar los datos introducidos por el usuario y llamar a otra que los envié adecuadamente al modelo para obtener ciertos resultados y mandárselos a la vista.

La implementación de una acción es casi automática y viene definida por el framework que implementa el controlador, para más detalles acudir a la documentación de StrutsCX **[STCX1]** y el código fuente.

Struts y StrutsCX (struts-1.1.jar y strutscx-0.9.5.jar) aportan:

- La implementación del núcleo del controlador: Formado por *com.cappuccinonet.strutscx.xslt.StrutsCXServlet* y otras clases.
- Las operaciones básicas entre Controlador y Sistema: Control de errores, interfaz de logado (log4j).
- Las operaciones básicas entre Controlador y Usuario: Validación de formularios.
- Las operaciones básicas entre Controlador y Modelo: Constructores de XML a partir de distintas fuentes (*DocumentBuilder*).
- Las operaciones básicas entre Controlador y Vista: Motor XSLT y minicontroladores para asociar los XSL correspondientes a cada acción.

3.1 Descriptor

Un descriptor es un fichero, normalmente XML, donde se *describe* el comportamiento de algún módulo de la aplicación y se indica la localización de los recursos necesarios.

La utilidad de estos ficheros es poder configurar desde fuera del código la aplicación, de forma que no sea necesario recompilar la aplicación para incluir nuevas páginas o modificar ligeramente el comportamiento.

Por ejemplo, se necesita descriptores para configurar el comportamiento del servidor de aplicaciones, el acceso al árbol JNDI (acceso a EJB, base de datos....), el flujo entre páginas.

Se puede ver el controlador, como un núcleo formado por varias capas de cebolla, la explicación de la capa del controlador esta dividida en función de los descriptores necesarios de la aplicación y siguiendo un orden lógico desde la capa exterior a la más interna.

4. Descriptor de la aplicación (web.xml)

Sirve para indicar al contenedor de aplicaciones (JBOSS) como ejecutar la aplicación.

El fichero web.xml esta alojado en la carpeta *COLDY\deploymentdescriptors\web* aunque para desplegar la aplicación correctamente hay que copiarlo a *COLDY\WEB-INF* como se indica en la especificación J2EE.

La estructura de este descriptor es común para todas las aplicaciones WEB basadas en J2EE y esta perfectamente documentada en la especificación **[WAPPDTD]**.

El descriptor utilizado esta incluido en el primer punto del apéndice A, cada bloque de etiquetas esta documentado y el lector no debería tener problemas para entenderlo; no obstante, se comenta los puntos particulares de la aplicación.

4.1 Listener

Un listener es una clase especial que se ejecuta automáticamente al desplegar la aplicación si ésta se configura adecuadamente.

Se ha configurado un listener para cargar y configurar el módulo que se ocupa de la autentificación: *edu.ucm.sip.inet.web.auth.listeners.InitializeLoginConfig.*

4.2 Filtros

Un filtro es una clase que se ejecuta por cada petición del usuario, una particularidad de los filtros es que pueden encadenarse formando una cadena. Para profundizar en la utilización de filtro se recomienda estudiar el patrón Intercep Filter **[PATTSUN].**

Se han configurado dos filtros:

- *edu.ucm.sip.inet.web.filters.SessionFilter*: Para asegurarse de que el usuario siempre tiene una sesión correctamente inicializada con el idioma y otros recursos.
- *edu.ucm.sip.inet.web.auth.filters.SignOnFilter*: Se encarga de comprobar que el usuario esta autentificado en todo momento.

4.3 Servlet

Los servlet son las clases que atienden a las peticiones del usuario. Para implementar la aplicación se ha utilizado el patrón MVC-2 (MVC2x), esto implica que un único servlet o jerarquía de servlet se encarga de realizar el control de la aplicación, manejando el flujo de páginas...

Aunque se han mejorado algunos detalles, la implementación del controlador y por tanto de los servlets, la realizan los frameworks utilizados, por lo tanto, tan solo tenemos que añadir en el descriptor el servlet que indica la documentación de StrutsCX **[STCX1]** **"***com.cappuccinonet.strutscx.xslt.StrutsCXServlet"*.

4.4 Errores

Aunque se ha dedicado un documento completo para explicar la gestión de errores del proyecto, se debe destacar que en el descriptor de la aplicación figura un punto importante de esta gestión.

El tag "<*error-page*>" indica al contenedor de aplicaciones qué hacer cuando se detecta un error o se lanza una excepción.

Un contenedor de aplicaciones como JBOSS puede indicar un error de dos formas distintas:

- Mediante un código de error, normalmente para errores del servidor o en la petición del usuario, por ejemplo: 400, 404, 500

Mediante el siguiente tag indicamos que en caso de producirse un error de tipo 500 dirija al usuario a la página errorpage.jsp.

```
<error-page>
        <error-code>500</error-code>
        <location>/web/pages/errorpage.jsp</location>
</error-page>
```

- Mediante una excepción, normalmente lanzada por la aplicación y no debidamente tratada, aunque también puede lanzarla el propio servidor de aplicaciones.

Mediante la siguiente etiqueta indicamos al contenedor que acción ejecutar, en caso de capturar una ServletException.

```
<error-page>
        <exception-type>javax.servlet.ServletException</exception-type>
        <location>/testEmpty.do</location>
</error-page>
```

4.5 Configuración

En el descriptor también figuran algunas etiquetas para indicar parámetros de configuración:

- *display-name* y *description*: Nombre y breve descripción de la aplicación.
- El parámetro de contexto: ***DSNAME*** que indica el nombre del datasource.
- El parámetro de contexto: **DMLOGIN** que indica el path del fichero de configuración del módulo autentificador.
- *session-timeout*: Tiempo en segundos, pasado el cual la sesión de usuario caduca sino ha ocurrido ningún evento.
- *welcome-file-list*: Página inicial.

5. Descriptor de Struts (struts-config.xml)

Sirve para indicar al servlet controlador el flujo entre páginas.

Aunque para desplegar la aplicación correctamente hay que copiar el descriptor a *COLDY\WEB-INF* como se indica en la documentación de Struts, durante el desarrollo de la aplicación se ha decidido alojar junto con el resto de los descriptores web en *COLDY\deploymentdescriptors\web.*

La estructura de este descriptor esta dividida en cuatro zonas especificadas con su correspondiente DTD **[STDTD].**

El descriptor utilizado esta incluido en el segundo punto del apéndice A, cada bloque de etiquetas esta documentado y el lector no debería tener problemas para entenderlo; no obstante, se comenta brevemente cada parte.

- **Formularios**: En esta parte se declaran los formularios utilizados por la aplicación. Los formularios son bean que recogen los datos de los formularios HTML.

  <form-bean name="logonForm"

  type="edu.ucm.sip.inet.web.auth.actions.LogonForm"/>.

- **Rutas globales**: Sirven para asociar un path con un alias, de forma que para hacer referencia a un path en los mapeos no halla que repetirlo constantemente.

  <forward name="welcome" path="/welcome.do"/>

- **Mapeos**: Es la parte fundamental de este fichero. Consiste en un conjunto de etiquetas *<action>* que asocian un path con una acción y definen las páginas (acciones) siguientes de acuerdo al resultado de la acción.

  <action path="/logon"

  type="edu.ucm.sip.inet.web.auth.actions.LogonAction"

  name="logonForm" scope="request" unknown="false"

  validate="false">

  <forward name="success" path="/welcome.do" redirect="false"/>

  <forward name="failed" path="/login.do" redirect="false"/>

  <forward name="error" path="/error.do" redirect="false"/>

  </action>

- Configuración de StrutsCX: En esta parte se especifican los recursos de la extensión StrutsCX.

  Path del fichero de configuración de la herramienta de logado Log4j y los ficheros de configuración de la herramienta para validar los datos de los formularios.

Para profundizar en la configuración de este descriptor se remite a la documentación de Struts.

6. Descriptor de StrutsCX (strutscx-config.xml)

Sirve para indicar configurar el funcionamiento de la extensión StrutsCX.

Para desplegar la aplicación correctamente hay que copiar este descriptor al path especificado en struts-config.xml, durante el desarrollo de la aplicación se aloja junto con el resto de los descriptores web en *COLDY\deploymentdescriptors\web.*

La estructura de este descriptor esta dividida en tres zonas especificadas con su correspondiente DTD **[STCXDTD].**

El descriptor utilizado esta incluido en el tercer punto del apéndice A, cada bloque de etiquetas esta documentado y el lector no debería tener problemas para entenderlo; no obstante, se comenta brevemente cada parte.

El framework StrutsCX aporta al controlador principalmente dos módulos:

- **DocumentBuilders**: Un conjunto de clases que se encargan de convertir en XML una serie objetos fuentes.

Se ha optado por extender la funcionalidad del DocumentBuilder por defecto *com.cappuccinonet.strutscx.xslt.StrutsCXStandardDocumentBuilder* con el nuevo builder *edu.ucm.sip.commons.strutscx.xslt.StrutsCXMsgErrorDocumentBuilder* para tener mayor control sobre los errores.

- **Transformers**: Un conjunto de clases que se encarga de transformar un XML en otro XML utilizando un motor de XSLT.

Se puede utilizar el transformador y el motor XSLT, sin embargo, se ha optado por mantener los que incorpora StrutsCX, ya que son eficientes y cumplen con toda la funcionalidad necesaria.

- **Módulos**: Aquí se especifica que clases se quieren utilizar para construir y transformar los XML.

```
<documentbuilder>
    <class id="standard">
    edu.ucm.sip.commons.strutscx.xslt.StrutsCXMsgErrorDocumentBuilder
    </class>
</documentbuilder>
<transformer>
    <class id="standard">
    com.cappuccinonet.strutscx.xslt.StrutsCXStandardTransformer
    </class>
</transformer>
```

- **Parámetros** Se pueden y deben inicializar ciertos parámetros para que el framework funcione correctamente:
  - Debugxml: Habilita la opción de visualizar el xml fuente, añadiendo a la petición un atributo *debugxml=true* a la query de la URL.
  - Lang y Encoding: Establece los idiomas y encodings que soporta la aplicación.
  - resources-properties: Indica el path del fichero de recursos internacionalizado, se establece uno por cada idioma soportado.

- **Mapeos**: Establece una relación biunívoca entre un alias y un fichero XSL, las acciones asociaran al XML que generan un XSL determinado utilizando estos alias.

```
<definition name="login">
        <put>/web/xsl/login.xsl</put>
</definition>
```

Para profundizar en la configuración de este descriptor nos remitimos a la documentación de StrutsCX.

7. Otros ficheros de configuración

Los framework utilizados necesitan de otros descriptores adicionales, para configurar determinados módulos como puede ser:

- Validator

El validator es un módulo desarrollado en los proyectos comunes de Jakarta, para más detalles sobre su funcionamiento leer la documentación de este módulo **[VAL1].**

Se utiliza para validar los datos introducidos por los usuarios en los formularios. Su utilización esta implícita en Struts y por ende en StrutsCX.

Los ficheros que se utilizan para configurar esta herramienta estan indicados en el fichero struts-config.xml:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames"
            value="/WEB-INF/validator-rules.xml, /WEB-INF/validation.xml"/>
</plug-in>
```

- Log4J

Log4J también es una herramienta desarrollada como proyecto de Jakarta e igualmente nos remitimos a la documentación oficial **[LOG4J1].**

Se utiliza para logar las trazas de error y debug de la aplicación.

Puesto que JBOSS también utiliza la misma herramienta de log, se ha decidido juntar el fichero de configuración de JBOSS y de la Intranet, por ello en el struts-config.xml se apunta al fichero de JBOSS:

```
<plug-in
    className="com.cappuccinonet.strutscx.util.StrutsCXLog4jInitPlugIn">
    <set-property property="config"
            value="JBOSS_HOME/server/coldy/conf/log4j.xml"/>
</plug-in>
```

El servidor de aplicaciones seleccionado JBOSS, también necesita sus propios descriptores, explicados en el correspondiente documento de herramientas utilizadas.

- jboss.xml
- jboss-web.xml
- ejb-jar.xml

Todos ellos se explican en el documento dedicado al servidor de aplicaciones.

Se ha adoptado la misma filosofía de utilizar ficheros XML como forma de configuración externa, para configurar los distintos módulos de los que se forma la intranet.

Estos recursos se alojan por defecto en la carpeta *resources* dentro de la carpeta principal de la aplicación web.

Por ejemplo, destacan:

- *login_datamodel.xml*: Que contiene un esquema del modelo de datos utilizado en el login, de forma que el módulo de autentificación vía JDBC, es prácticamente independiente del modelo de datos.
- pwd_rules.xml: Indica las reglas que deben superar las password de los usuarios dados de alta en la aplicación.

Bibliografía

[WEBSA]
http://www.osmosislatina.com/aplicaciones/servidor_web.htm

[PATTSUN]
http://java.sun.com/blueprints/corej2eepatterns/index.html
[PATTPC]
http://www.programacion.net/tutorial/patrones/

[STRUTS]
http://jakarta.apache.org/struts/
[STCX1]
http://it.cappuccinonet.com/strutscx/index.php
[STPC1]
http://www.programacion.com/tutorial/struts/
[STPC2]
http://www.programacion.net/articulo/tips_struts/

[WAPPDTD]
web-app_2_3.dtd
[STDTD]
struts-config_1_1.dtd
[STCXDTD]
strutscx-config_094.dtd

[VAL1]
http://jakarta.apache.org/commons/validator/
[LOG4J1]
http://logging.apache.org/log4j/docs/

CONTROL DE VERSIONES

1.0 → Estructura y conceptos básicos.
1.1 → Descriptores básicos.
2.0 → Descriptores del framework.
2.1→ Referencias a la gestión de errores y apéndices.
2.2 → Apéndices

APÉNDICE 1 - Descriptores de Aplicación

1. application.xml y web.xml

El application.xml aunque no ha sido comentado es otro descriptor común para todas las aplicaciones web basadas en J2EE que indica al servidor de aplicaciones datos como el path y nombre de la aplicación.

```
<application>
    <display-name>COLDY</display-name>
    <module>
            <web>
            <web-uri>coldy.war</web-uri>
            <context-root>/coldy</context-root>
            </web>
    </module>

    <module>
            <ejb>coldy.jar</ejb>
    </module>
</application>
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd">

<!-- Descriptor de la aplicacion -->
<web-app>

   <display-name>INTRANET DSIP CX</display-name>
   <description><![CDATA[Intranet con StrutsCX]]></description>

   <context-param>
      <param-name>DSNAME</param-name>
      <param-value>java:/OracleDS</param-value>
      <description>
         Nombre del Datasource utilizado para obtener conexiones de base de datos
      </description>
   </context-param>
```

```
<context-param>
   <param-name>DMLOGIN</param-name>
   <param-value>/resources/login_datamodel.xml</param-value>
   <description>Configuracion del modelo de datos del login</description>
</context-param>

<!-- Filter Configuration -->
<filter>
   <filter-name>SessionFilter</filter-name>
   <filter-class>edu.ucm.sip.inet.web.filters.SessionFilter</filter-class>
</filter>

<filter>
   <filter-name>SignOnFilter</filter-name>
   <filter-class>edu.ucm.sip.inet.web.auth.filters.SignOnFilter</filter-class>
</filter>

<filter-mapping>
   <filter-name>SessionFilter</filter-name>
   <url-pattern>*.do</url-pattern>
</filter-mapping>

<filter-mapping>
   <filter-name>SignOnFilter</filter-name>
   <url-pattern>*.do</url-pattern>
</filter-mapping>

<listener>
   <listener-class>
         edu.ucm.sip.inet.web.auth.listeners.InitializeLoginConfig
  </listener-class>
</listener>
```

```
<!-- Configuracion estandar (con debug) del Action Servlet (Struts Controller) -->
<servlet>
   <servlet-name>action</servlet-name>
   <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
   <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
   </init-param>
   <init-param>
      <param-name>debug</param-name>
      <param-value>5</param-value>
   </init-param>
   <init-param>
      <param-name>detail</param-name>
      <param-value>5</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>
<servlet>
   <servlet-name>StrutsCXServlet</servlet-name>
   <servlet-class>com.cappuccinonet.strutscx.xslt.StrutsCXServlet</servlet-class>
   <init-param>
      <param-name>debug</param-name>
      <param-value>false</param-value>
   </init-param>
   <load-on-startup>2</load-on-startup>
</servlet>

<!-- Mapping del Controlador, tratamos cualquier URL que termine en .do -->
<servlet-mapping>
   <servlet-name>action</servlet-name>
   <url-pattern>*.do</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>StrutsCXServlet</servlet-name>
   <url-pattern>/StrutsCXServlet</url-pattern>
</servlet-mapping>
```

```
    <session-config>
        <session-timeout>60</session-timeout>
    </session-config>
    <!-- Página de bienvenida -->
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <error-page>
        <error-code>500</error-code>
        <location>/web/pages/errorpage.jsp</location>
    </error-page>

    <error-page>
        <exception-type>javax.servlet.ServletException</exception-type>
        <location>/error.do</location>
    </error-page>

</web-app>
```

2. struts-config.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config
 PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
 "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

   <form-beans>
      <form-bean name="logonForm"
         type="edu.ucm.sip.inet.web.auth.actions.LogonForm"/>
   </form-beans>

   <global-forwards>
      <forward name="welcome" path="/welcome.do"/>
   </global-forwards>

   <!-- Definiciones de los Action Mappings -->
   <action-mappings>

      <!-- Entrada de datos para el login -->
      <action
         path="/testError"
         type="edu.ucm.sip.inet.web.actions.TestError">
         <forward name="success" path="/StrutsCXServlet"/>
         <forward name="error" path="/testEmpty.do?debugxml=true"/>
      </action>

      <!-- Pagina principal -->
      <action
         path="/welcome" type="edu.ucm.sip.inet.web.actions.WelcomeAction">
         <forward name="success" path="/StrutsCXServlet"/>
      </action>

      <!-- Entrada de datos para el login -->
      <action
         path="/login"
         type="edu.ucm.sip.inet.web.auth.actions.LoginAction">
         <forward name="success" path="/StrutsCXServlet"/>
      </action>
```

```
<!-- Action que se encarga de validar la session del usuario -->
<action
   path="/logon"
   type="edu.ucm.sip.inet.web.auth.actions.LogonAction"
   name="logonForm"
   scope="request"
   unknown="false"
   validate="false"
   >
   <forward
      name="success"
      path="/welcome.do"
      redirect="false"
      />
   <forward
      name="failed"
      path="/login.do?debugxml=true"
      redirect="false"
      />
   <forward
      name="error"
      path="/login.do"
      redirect="false"
      />
</action>

<action
   path="/Logoff"
   type="edu.ucm.sip.inet.web.auth.actions.LogoffAction">
   <forward
      name="success"
      path="/login.do"/>
</action>
```

```
<!--Muestra los resultados1 del listado personalizado del PERSONAL-->
<action
   path="/listadoStaff"
   type="edu.ucm.sip.inet.web.actions.staff.ListadoStaff">
   <forward name="success" path="/StrutsCXServlet"/>
</action>

<!--Muestra las opciones del listado personalizado de PERSONAL-->
<action
   path="/listado"
   type="edu.ucm.sip.inet.web.actions.staff.Listado">
   <forward name="success" path="/StrutsCXServlet"/>
</action>

<!--Muestra la pantalla para insertar un persona del PERSONAL-->
<action
   path="/addStaff"
   type="edu.ucm.sip.inet.web.actions.staff.AddStaff">
   <forward name="success" path="/StrutsCXServlet"/>
</action>

<!--Muestra la pantalla de resultado después de la inserción del PERSONAL-->
<action
   path="/addStaffResul"
   type="edu.ucm.sip.inet.web.actions.staff.AddStaffResul">
   <forward name="success" path="/StrutsCXServlet"/>
</action>

<!--Muestra la pantalla para recoger el NIF y posteriormente el PERSONAL-->
<action
   path="/mostrarStaff"
   type="edu.ucm.sip.inet.web.actions.staff.MostrarStaff">
   <forward name="success" path="/StrutsCXServlet"/>
</action>
```

```
        <!--Muestra la pantalla con los datos del PERSONAL-->
        <action
            path="/staffResul"
            type="edu.ucm.sip.inet.web.actions.staff.StaffResul">
            <forward name="success" path="/StrutsCXServlet"/>
        </action>

    </action-mappings>

    <message-resources parameter="ApplicationResources" null="false"/>

    <!-- StrutsCXLog4jInitPlugIn -->
    <plug-in className="com.cappuccinonet.strutscx.util.StrutsCXLog4jInitPlugIn">
        <set-property property="config"
            value="E:/jboss-3.2.3/server/coldy/conf/log4j.xml"/>
    </plug-in>

    <!-- StrutsCXPlugIn -->
    <plug-in className="com.cappuccinonet.strutscx.util.StrutsCXPlugIn">
        <set-property property="config"
            value="/WEB-INF/strutscx-config.xml"/>
    </plug-in>


    <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
        <set-property property="pathnames" value="/WEB-INF/validator-rules.xml,
                                    /WEB-INF/validation.xml"/>
    </plug-in>

</struts-config>
```

3. strutscx-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
   <documentbuilder>
      <class id="standard">
      edu.ucm.sip.commons.strutscx.xslt.StrutsCXMsgErrorDocumentBuilder
      </class>
   </documentbuilder>

  <transformer>
      <class id="standard">
      com.cappuccinonet.strutscx.xslt.StrutsCXStandardTransformer
      </class>
   </transformer>

   <debugxml>
      <on>ON</on>
   </debugxml>

   <lang>en es</lang>
   <encoding>
      <en>ISO-8859-1</en>
      <es>ISO-8859-1</es>
   </encoding>

   <resources-properties>
      <on>ON</on>
      <en>/WEB-INF/xml/variables_en.xml</en>
      <es>/WEB-INF/xml/variables_es.xml</es>
   </resources-properties>
```

```
   <strutscx-definitions>
      <definition name="emptyXSL">
         <put>/web/xsl/empty.xsl</put>
      </definition>

       <definition name="welcome.intro">
         <put>/web/xsl/welcome.xsl</put>
      </definition>

      <definition name="login">
         <put>/web/xsl/login.xsl</put>
      </definition>

      <definition name="listado">
               <put>/web/xsl/listado.xsl</put>
      </definition>

       <definition name="listadoStaff">
               <put>/web/xsl/listadoStaff.xsl</put>
       </definition>

       <definition name="addStaff">
               <put>/web/xsl/addStaff.xsl</put>
      </definition>

       <definition name="addStaffResul">
               <put>/web/xsl/addStaffResul.xsl</put>
      </definition>

      <definition name="mostrarStaff">
               <put>/web/xsl/mostrarStaff.xsl</put>
      </definition>

      <definition name="staffResul">
               <put>/web/xsl/staffResul.xsl</put>
      </definition>
   </strutscx-definitions>
</properties>
```

4. validation.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<form-validation>
  <formset>
    <form name="logonForm">
      <field property="username"
        depends="required,mask,maxlength">
        <msg name="mask" key="login.mask.error"/>
        <msg name="maxlength" key="login.username.maxlength"/>
        <var>
          <var-name>mask</var-name>
          <var-value>^[a-zA-Z0-9]*$</var-value>
        </var>
        <var>
          <var-name>maxlength</var-name>
          <var-value>10</var-value>
        </var>
      </field>
      <field property="password"
        depends="required,minlength,maxlength">
        <msg name="minlength" key="login.password.minlength"
          resource="false"/>
        <msg name="maxlength" key="login.password.maxlength"
          resource="false"/>
        <var>
          <var-name>minlength</var-name>
          <var-value>5</var-value>
        </var>
        <var>
          <var-name>maxlength</var-name>
          <var-value>10</var-value>
        </var>
      </field>
    </form>
  </formset>
</form-validation>
```

5. log4j.xml

Por ejemplo se puede añadir algo como

```
 <appender name="ROLLY" class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="${jboss.server.home.dir}/log/coldy_hist.log"/>
    <param name="Append" value="true"/>
    <param name="MaxFileSize" value="100KB"/>
    <param name="MaxBackupIndex" value="1"/>
    <layout class="org.apache.log4j.PatternLayout">
       <param name="ConversionPattern" value="%d %-5p - [%C{1}] %m%n"/>
    </layout>
 </appender>

<category name="org.apache.commons">
            <priority value="INFO"/>
</category>

<category name="org.jboss">
            <priority value="INFO"/>
</category>

   <logger name="com.cappuccinonet.strutscx">
            <level value="INFO"/>
   </logger>

   <logger name="edu.ucm.sip.inet.web.actions">
            <level value="DEBUG"/>
   </logger>

<root>
   <appender-ref ref="CONSOLE"/>
   <appender-ref ref="ROLLY"/>
</root>
```

CAPA VISTA

Intranet para un Dpto. Universitario

Conceptos y Desarrollo de la Capa de Presentación

2 de Junio de 2004, v2.2

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

1. PRÓLOGO

A lo largo de este documento se ha procurado explicar cada uno de los conceptos básicos necesarios para comprender el resto del documento, no obstante es recomendable que el lector tenga ciertos conocimientos sobre el diseño web (HTML...)

Para comprender la totalidad del documento es imprescindible estar familiarizado con el patrón MVC por lo que se recomienda leer el "Informe Tecnológico" antes que este documento.

2. INTRODUCCIÓN

En este documento se detalla el proceso completo que se ha seguido para desarrollar la capa de presentación o vista[1].

Se ha recopilado el análisis, diseño e implementación de la capa en un único documento con el objeto de facilitar el mantenimiento de las pantallas de la aplicación, por ello se hace especial énfasis en explicar la solución adoptada,.

El documento esta dividido en cuatro partes principales:

1.- La primera parte (punto 3) es una breve relación de los *conceptos* más importantes, necesarios para comprender el documento.

2.- En la segunda parte (punto 4) se realiza una introducción de cada una de las *alternativas* contempladas para desarrollar la interfaz del usuario y una breve discusión de porque se ha escogido una de ella en concreto.

3.- La tercera parte (punto 5) es la más importante a la hora de comprender el *diseño* e implementación de esta capa, ya que se explica *opción* elegida y como se ha utilizado.

4.- La última parte (punto 6) es una guía para el *desarrollador* y una recopilación de consejos técnicos con el objeto de facilitar el mantenimiento de las pantallas actuales y ayudar en el desarrollo de nuevas funcionalidades.

[1] La capa Vista fue introducida en el Informe tecnológico cuando se describe el MVC.

3. Definiciones y conceptos básicos

3.1 Contenidos estáticos y dinámicos

El patrón *MVC* esta formado por tres capas Modelo, Vista y Controlador **[PATTSUN]**.

La capa vista esta formada por la parte de la aplicación que se encarga de generar y presentar el interfaz al usuario.

Podemos clasificar dos tipos de interfaz en función de su contenido:

- **Estáticos**, solo ofrecen contenido estático, no varía, aunque puede ser actualizado.
- **Dinámicos,** se dice que el contenido es dinámico si cambia con el tiempo o ante eventos del usuario.

Un interfaz estático o dinámico puede **estar construido previamente**, como por ejemplo: una imagen bmp (estático), un gif animado (dinámico) o **ser construida en función de parámetros dinámicos**, como peticiones del usuario, contexto de la aplicación... hablamos entonces de **interfaces generados dinámicamente**.

En el contexto de aplicaciones web, el resultado de la capa vista suelen ser documentos *HTML estáticos generados dinámicamente* en función del contexto de la aplicación.

HTML es la forma más extendida de codificar la información web, aunque no es el único formato, también abundan en XML, PDF, MSWord, Flash...

3.2 Generación Dinámica de la vista en Aplicaciones Web

Se dice que la información (estática o dinámica) puede ser generada **dinámicamente** en dos sitios:

- **Cliente**: En el propio navegador, pro ejemplo: mediante JavaScript, VBScript...
- **Servidor**: En el servidor, por ejemplo mediante: Servlets, JSP, ASP...

No debemos confundir la generación dinámica de documentos en cliente con documentos dinámicos.

Llegados a este punto es necesario tener los conceptos de información dinámica y generación dinámica bien claros. Aún para gente familiarizada con el tema esto puede ser confuso. Trataremos de aclararlo con un último ejemplo más técnico.

Por ejemplo: Consideremos un script implementado con JavaScript y contenido en una página HTML. Si el script se encarga de recorrer el árbol dom del documento HTML y modificarlo añadiendo nuevos párrafos. Se entiende que es generación dinámica en cliente; sin embargo, si el script, tan solo provoca la aparición de una imagen cuando el ratón pasa por cierta zona del documento, se entiende como documento dinámico (que ha podido ser generado dinámicamente o no), tanto en servidor o como en cliente.

3.3 Contenido, Continente y Formato

Como ya se ha mencionado antes hay que tener en cuenta que la información presentada por una aplicación web esta sujeta a las mismas consideraciones que la información impresa en papel...

Cuando hablamos del **contenido** de la información, nos referimos a la información en si misma a los datos calculados por la aplicación.

Si tenemos en cuenta que la información siempre es digital, podemos referirnos al **continente** , no como en que dispositivo esta almacenada (disco duro, CD-Rom) o como esta organizada (base de datos, fichero de texto), sino al tipo de documento donde esta almacenada la información (html, pdf...).

Por último, el **formato** es como se ve la información en última instancia, en negrita, de color azul... al margen de estar un pdf o en un página HTML.

3.4 Formateo de contenidos mediante transformaciones

Este apartado no es muy extenso puesto que se sale de los objetivos del documento, de hecho no es más que una introducción del siguiente punto. Por otro lado, tampoco se explican en detalle conceptos fundamentales a los que se hace referencia, por lo que es necesario estar familiarizado con el tema.

Recomendamos encarecidamente la lectura del tutorial [**XSLPC01**] y sus "*hermanos*" tanto para iniciarse, como para entrar en detalle.

- XML, HTML y XSLT

El lenguaje HTML fue especialmente diseñado para dar formato a los datos de una forma fácil y eficiente. Un documento HTML mantiene los datos, junto con marcas que determinan el formato[2] y aunque es un lenguaje muy versátil para formatear información este es su mayor inconveniente[3].

En cambio, el lenguaje XML se creo específicamente para representar información, solo datos **[XMLC01]**.

Por lo tanto, si generamos la información contenida en documentos XML y después podemos transformarla de algún modo en documentos HTML para que quede formateada tendremos resueltos muchos problemas.

Con este objetivo se concibió la especificación de XSL, una serie de reglas para implementar motores de transformación que generen documentos a partir de otros documentos en XML.

[2] Este hecho da lugar a tanta confusión cuando se distingue entre contenido, continente y formato hablando de páginas web.

[3] No es propósito de este documento explicar porque es un inconveniente no tener los datos aislados.

Transformación mediante plantillas XSLT

XSLT es un subconjunto de XSL que se centra en la transformación de XML en XML. En concreto se suele aplicar para transformar información en XML en XHTML, que es un subconjunto de XML con las mismas ventajas que HTML **[XMLPC04]**.

La transformación de XML a XHTML necesita un motor de transformación XSL que puede ejecutarse en el servidor (.: Xalan) o en cliente (.: MSXML – IE > 5.0). Es obvio, que esta transformación requiere tiempo y por lo tanto, añade una perdida de rendimiento en la presentación de la información que según el caso puede ser considerable.

En resumen, la solución adoptada para este proyecto consiste a grandes rasgos:

- Obtención de información de base de datos y ficheros, a partir de solicitudes del usuario.
- Generación dinámica de XML con la información recogida.

Transformar los XML en XHTML mediante plantillas XSLT.

4. Generación y Transformación de XML

En adelante, nos centraremos en como generar XML y como transformarlo mediante plantillas XSLT **[MERC01]**.

J2EE especifica dos tecnologías **[JSPPC01]** para generar información en un navegador:

1.- SERVLETS

Los servlet puede escribir código ASCII en el cliente directamente, utilizando esta característica podemos generar la vista en forma de documentos XML y aplicando la transformación en cliente obtener HTML.

Utilizando los métodos getWriter o getOutputStream conseguimos objetos en los que podemos escribir directamente código ASCII en el cliente.

```
PrintWriter pw = response.getWriter();
pw.print("<?xml-stylesheet href="introuserhtml.xsl" type="text/xsl"?>");
pw.print("<user><name>Pepe</name ><pwd>XXX</pwd ></user >")
```

Podemos incluir este tipo de código en los métodos doGet, doPost... o aislarlo en un método independiente al estilo.

```
…
pw.print("<?xml-stylesheet href="introuserhtml.xsl" type="text/xsl"?>");
printUser(pw,user);
void printUser(PrintWriter pw, User user) {
    if (pw!=null) {
            pw.print("<user>");
            pw.print("<name>"+user.getName()+"</name>");
            pw.print("<pwd>+user.getPwd()+"</pwd >");
            pw.print("</user >")
        }
}
```

Siguiendo una metodología más orientada a objetos podemos traspasar al objeto user (generalmente un JavaBean) la responsabilidad de cómo "pintarse" o presentarse. xml y javabeans

```
public String toXML() {
        StringBuffer sb = new StringBuffer(user.size());
            sb.append("<user>");
            sb.append("<name>"+user.getName()+"</name>");
            sb.append("<pwd>+user.getPwd()+"</pwd >");
        sb.append("</user >")
        return sb.toString();
}
….
```

pw.print(“<?xml-stylesheet href="introuserhtml.xsl" type="text/xsl"?>”);

pw.print(user.toXML());

Si como en este caso estamos utilizando XML con intención de utilizar XSLT y no HTML directamente podemos utilizar estándares JAXP, por ejemplo: un objeto Document de JDOM para representar User en XML y no un String [**XMLPC02**].

Si no queremos usar transformación en cliente, debemos utilizar un API que nos permita utilizar un transformador en el servidor. Todos funcionan de forma semejante.

El proceso para la transformación en server consta de tres fases.

1) Cargar una fuente de datos con el XML. Normalmente datos de memoria en forma de array de bytes o semejante.
2) Cargar una fuente de datos con la plantilla XSL, normalmente a partir de un fichero. xslPath= “resources/xsl/ introuserhtml.xsl”
3) Ejecutar la transformación con las fuentes de datos cargadas y obtener el resultado. El resultado de la transformación suele ser un OutPutStream
4) Asociar el resultado (HTML) a la salida que va al cliente igual que hemos hecho antes con XML y transformación en cliente.

Veamos un ejemplo utilizando el transformador de Jakarta.Apache Xalan

```
1)
ByteArrayOutputStream  baos = new ByteArrayOutputStream();
PrintWriter printWriter = new PrintWriter(baos);
pw.print(user.toXML());
final byte[] convertedCharArray =
        new String(baos.toByteArray(),"UTF-8").
                getBytes(defaultCharsetName);
Source xmlSource= new StreamSource(
        new ByteArrayInputStream(convertedCharArray));
2)
String xslPath= “./resources/xsl/introuserhtml.xsl”
Source xslSource = new StreamSource(xslPath);
3)
final PrintWriter out = response.getWriter();
Transformer transformer = TransformerFactory.newInstance().
                                    newTransformer(xsl);
4)
transformer.transform(xml, new StreamResult(out));
out.flush();
```

2.- JSPs

El código JSP se visualiza directamente en el navegador, luego si escribimos el contenido XML/HTML directamente tendremos la pantalla implementada [**XMLPC03**].

En los JSP podemos incluir código java para ayudarnos a generar el contenido, incluso podemos llamar a métodos que realicen la transformación XSL en el servidor.... De hecho cuando el servidor de aplicaciones compila los JSPs genera servlets así que tenemos la misma potencia.

Para tareas habitúales como realizar la transformación o recorrer una colección e generando el XML de cada elemento se suelen usar etiquetas personalizadas o taglib (librerías de etiquetas).

Los JSPs permiten ser construidos en árbol mediante includes de otros JSPs esto nos permite por un lado "trocear" las vistas por zonas y asociarlas a determinadas condiciones como permisos... y por otro modularidad frente al modelo y la presentación.

```
<jsp:useBean id="user" type="com.app.User" scope="session"/>
<%@include file="view/component/head.jsp"%>
<%@include file="view/component/user/xmlUser.jsp"%>
<%@include file="view/component/foot.jsp"%>
```

donde xmlUser.jsp tiene esta pinta

```
<user>
    <name><jsp:getProperty name="user" property="name"></name>
    <pwd><jsp:getProperty name="user" property="pwd"></pwd>
</user>
```

4.1 Decoradores

Crear el XML de cada vista utilizando beans que "saben" como representarse en XML facilita la creación de plantillas XSL ya que eliminamos casos particulares. En los ejemplos anteriores sabes que un usuario tiene dos propiedades name y pwd.

Cuando decimos que los beans saben representarse en XML es porque tienen métodos como toXML... sin embargo estos método tiene una serie de incovenientes:

- No son estándares
- Mucha variedad: String toXML(); void toXML(PrintWriter pw)...
- Si cambia la representación necesitamos recompilar el módelo.
- En cierto modo hace a la vista dependiente del modelo.
- Varias aplicaciones pueden compartir el mismo bean, pero desear distinto formato XML.

Cuando vimos como utilizar los JSP en la capa Vista, hemos utilizamos un JSP por vista y otro por bean, aprovechándonos de la directiva include. Podemos abstraer esta forma de trabajar y crear plantillas de XML, en adelante decoradores.

Un decorador es un fichero en formato XML a modo de plantilla que define como es un bean en formato XML, podemos darle la extensión XMLT (XML template). Por ejemplo

user.xmlt sería algo como

```
<user>
    <name>
    {user.name}
    </name>
    <pwd>
    {user.pwd}
    </pwd>
</user>
```

Este mecanismo necesita una infraestructura java para poder asociar a cada bean su plantilla y usarlas.

4.2 Alternativas analizadas

En este punto se ha tratado de realizar un estudio comparativo de las alternativas analizadas para tratar de concluir porque se escogió una en concreto.

A continuación se enumeran las siete alternativas estudiadas.

- Servlets + JSPs
- JSPs + TagLibs
- Framework
    - Struts **[STRUTS]**
        - Stxx **[STXX1]**
        - **StrutsCX [STCX1].**
    - Spring

Después de implementar pruebas con todas ellas y al final se escogió StrutsCX.

Las dos primeras opciones (Servlets + JSPs y JSPs + TagLibs) requieren implementar la lógica del controlador además de la vista y el modelo **[SUNWEB01]**.

Si se utiliza un framework como Struts el controlador solo hay que configurarlo y basta con implementar el modelo y definir la vista, además se dispone de un entorno con tratamiento de errores, facilidades de logado... **[STPC3]**.

No obstante es complicado utilizar plantillas XSL en el framework de Struts ya que gran parte de sus ventajas, por no decir todas, están basadas en taglibs **[STPC2]** que generan HTML, sin embargo existen dos extensiones de Struts que soportan XSL (Stxx y StrutsCX).

Entre Stxx y StrutsCX no hay mucha diferencia, ambos son framework gratuitos basados en Struts. Para decantarnos por StrutsCX se tuvo en cuenta la opinión de expertos **[STCX2]** y foros de la Red **[STCX3]**, el mantenimiento de ambos frameworks, el número de bugs, documentación... en concreto el número de ejemplos encontrados para StrutsCX fue lo que inclino la balanza del lado de StrutsCX.

En el momento de implementar el proyecto tanto los proyectos Struts, Stxx y StrutsCX estaban bastante parados, ya que otro framework Spring **[SPR01]** con mejores expectativas en cuanto a rendimiento y flexibilidad esta imponiéndose en el mercado, esta recomendado por destacados fabricantes de software: Borland, BEA (weblogic), IBM (was).

Spring no se ha considerado por tres motivos **[SPR02]**:

1.- Cuando se comenzó este proyecto, Spring solo estaba disponible en versión betta y apenas tenía documentación, en estos momentos el proyecto esta demasiado avanzado como para plantearse otro framework.

2.- Ninguno de los integrantes del grupo tiene experiencia con este proyecto.

3.- En la documentación de Spring se menciona que es relativamente sencillo integrar Struts con Spring, facilitando la migración de Struts a Spring.

Ya que Spring no impide el uso de Struts y costaría demasiado tiempo integrarlo se postpone su integración, aunque se recomienda.

5. El framework: Struts y StrutsCX

5.1 Introducción

La Intranet como cualquier otra aplicación WEB necesita de un sistema que muestre al usuario los datos, este sistema se denominado Capa Vista.

El funcionamiento implementado para la capa vista de la Intranet es muy simple de exponer:

La información solicitada por el usuario se genera dinámica y es representada por un XML, dicha información es transformada mediante plantillas XSL por un motor XSLT de tal forma que al usuario se le presenta la información, ya transformada, en HTML.

Sin embargo, no es tan fácil de implementar **[ORB01]**:

Hay que pensar un formato para los XML, idear una forma flexible para validar la información introducida por el usuario, integrar un motor de XSLT...

Con la intención de no "*reinventar la rueda"* se ha optado por utilizar un framework, StrutsCX, basado en el archí conocido Struts.

5.2 Aportaciones StrutsCX

StrutsCX permite al programador trabajar cómodamente con la vista. El desarrollador puede centrarse en el diseño de las páginas, implementando solo pequeñas acciones.

El framework realiza las siguientes tareas **[STCX1]**:

- Mediante los DocumentBuilder se pueden generar XML automáticamente, a partir de información residente en objetos java. Esto reduce el trabajo al implementar la lógica de negocio.
- Añade al XML fuente toda la información del contexto web. Además incluye la información necesario para gestionar errores del usuario y de la aplicación.
- Los documentos XML generados tienen una estructura determinada, lo que facilita la tarea de construir las plantillas XSL.
- Implementa un motor de transformación XSLT propio StrutsCXTransformer, aunque se puede sustituir por cualquier otro externo (Xalan...).

También incluye utilidades para **[STCX2]**:

- Depurar los XML generados.
- Internacionalizar los datos.
- Validar formularios.
- Realizar la transformación XSL en cliente.

5.3 Estructura de un XML

Los documentos XML que genera StrutsCX tienen una estructura determinada que explicaremos basándonos en el ejemplo expuesto en el Apéndice 1.

"root", Tag Principal

Un documento XML válido solo puede tener una etiqueta principal, StrutsCX siempre nombre a esta etiqueta ***root***, lo que facilita la arquitectura de las plantillas.

El documento se divide en siete zonas que corresponde con la serie de etiquetas que genera el framework, es importante destacar que el nombre de las etiquetas no es modificable, su contenido esta bien definido y no todas son obligatorias.

A continuación se explica cada una de ellas:

1. "data". Datos generados por la acción

Zona reservada a la información generada por la acción ya sea dinámica o estática.

En ella se recoge el **contenido de la página**. Se suele generar en función del idioma, incluso de otros parámetros como el encoding, user-agent (navegador), rol del usuario...

StrutsCX soporta dos mecanismos para incluir esta información:

- Introducir en la request un objeto DOM que representa el XML:
  - org.w3c.dom.Documents
  - org.jdom.Documents
  - org.dom4j.Documents
- Transformación automática de objetos java (bean, EJBs...) a XML mediante un Builder. Un Builder[4] transforma algunas instancias de clases Java (Beans, Vector)... en XML, rigiendose por reglas como: "*cada atributo del bean es un tag del XML, cuyo contenido es el valor del atributo como String*".

El programador es libre de incluir el contenido que desee en este tag *data.* Puede incluir desde objetos suministrados por los EJBs que conforman el modelo, hasta ficheros con información estática.

El contenido del ejemplo sirve para facilitar la internacionalización de los mensajes de la pantalla de login, se ha obtenido a partir de un fichero XML, en función el idioma.

[4] El programador puede utilizar StrutsCXDocumentBuilder o implementarse los propios.

El documento original es "*login_es.xml*"

```
<messages>
    <titlewin>Welcome to</titlewin>
        ...
</messages>
```

Y para carga información estática en el XML fuente, se hace en cuatro pasos como se explica con el siguiente fragmento de código:

1.- Se localiza el fichero según los parámetros que queramos.

```
String xml_file = "/WEB-INF/xml/login_" + getLocale(request).getLanguage() + ".xml";
```

2- Se carga en memoria a través de un buffer para optimizar los proceso de I/O.

```
StrutsCXXMLReader xmlReader = new StrutsCXXMLReader(servlet,xml_file);
```

3.- Se crea un objeto adecuado *org.jdom.Documents.*

```
Document docMsgs = xmlReader.getDocument();
```

4.- Se guarda en la petición para que el DocumentBuilder lo incluya en el XML que llega al cliente.

```
request.setAttribute("login_file", docMsgs);
```

2. "resourceproperties" Información de los recursos de mensajes (i18n)

Reservada para los mensajes internacionalizados, si estos mensajes son muchos y particulares de una sola página se recomienda incluirlos en el tag data y utilizar este para los mensajes comunes a todas las pantallas.

El contenido de esta zona se determina a partir de la configuración de StrutsCX, en el siguiente fragmento obtenido del fichero strutscx-config.xml.

```
<resources-properties>
  <on>ON</on>
  <en>/WEB-INF/xml/variables_en.xml</en>
  <es>/WEB-INF/xml/variables_es.xml</es>
</resources-properties>
```

En nuestro ejemplo se ha cargado el contenido del fichero variables_es.xml

```
<resourcesproperties lang="en">
  <lastupdate>May 1, 2004</lastupdate>
  <lastupdatetext>Last update</lastupdatetext>
  <ucm>Complutense University of Madrid</ucm>
  ...
  <error>
    <text>CappuccinoNet - Ooops!</text>
  </error>
</resourcesproperties>
```

Es el mecanismo que ofrece StrutsCX para implementar la internacionalización.

El programador debe ser cuidadoso con este recurso ya que puede crecer rápidamente y degradar el rendimiento de la transformación.

3. "actionforms" Contenido de los formularios (opcional)

Este tag es incluido automáticamente por el framework en el caso de que exista un ActionForm en la acción.

```
<actionform>
    <LogonForm>
        <class>...auth.actions.LogonForm</class>
        <multipartRequestHandler />
        <page>0</page>
        <password />
        <resultValueMap />
        <ActionServletWrapper>
            <class>...action.ActionServletWrapper</class>
        </ActionServletWrapper>
        <username />
        <validatorResults />
    </LogonForm>
</actionform>
```

El programador no puede (o no debe) alterar el contenido de este tag, pero si puede y debe beneficiarse de el, normalmente esta información se utiliza para validar mediante JavaScript los datos introducidos por el usuario.

4. "errors" Información de errores (opcional)

El framework genera este tag cuando el *Validator* asociado a un *ActionForm* encuentra un error.

La configuración de los validadores utilizados se indican el archivo de configuración de Struts:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames" value="/WEB-INF/validator-
    rules.xml, /WEB-INF/validation.xml" />
</plug-in>
```

- *validator-rules.xml* contiene la configuración del comportamiento del validador
- *validation.xml* informa sobre el validador que tiene asociado cada formulario.

Dentro del tag *"<errors>"* se encuentra un tag con el nombre de cada campo que ha fallado, en nuestro ejemplo: la password es nula o no contiene entre 5 y 150 caracteres.

```
<errors>
    <password />
</errors>
```

En principio, este tag como el anterior es transparente para el programador, sin embargo, se recomienda extender esta funcionalidad para expresar más información del error.

```
<errors>
    <password >inferior a 5 carácteres</password >
</errors>
```

Una buena plantilla XSL debe cerciorarse que no existe este tag y si lo encuentra actuar en consecuencia, por ejemplo: destacando los errores.

5. "request" Información de la Request

En esta zona el framework incluye todos los atributos y parámetros de la petición del usuario, es decir, contiene todos los atributos y parámetros contenidos en el objeto ServletRequest.

```
<request>
  <xsl_key>login</xsl_key>
  <xsl_clientside>false</xsl_clientside>
  <encoding_key>ISO-8859-1</encoding_key>
  ...
  <debugxml>true</debugxml>
  <password />
  <username>113</username>
  ...
</request>
```

Parte de esta información es incluida por el propio entorno y utilizada por el motor de transformación de StrutCX, por eso el programador no debería borrar ningún atributo/parámetro que no fuese suyo, aunque es libre de incluir los que considere necesarios.

6. "session". Información de la Sesión

Aquí se incluyen todos los parámetros y atributos contenidos en la sesión del usuario.

```
<session>
  <jsessionid>C16577B491E...</jsessionid>
  <com.cappuccinonet.strutscx.domainparams_flag>true
    </com.cappuccinonet.strutscx.domainparams_flag>
  <date_time_today>
      <string>Sun May 23 19:29:54 CEST 2004</string>
        ...
  </date_time_today>
  ...
</session>
```

Esta información a podido ser incluida por el servidor de aplicaciones[5], por el framework Struts, la extensión StrutsCX o por la propia aplicación, lo que implica las mismas restricciones al usuario que la sección anterior.

Esta información nos ayuda a conseguir el path de más información como imágenes... Nos muestra información del usuario nombre, fecha...

La forma de utilizar StrutsCX esta perfectamente explicada en la documentación **[STCX1] [STCX]**, por lo que, en adelante nos centraremos en explicar la arquitectura de las plantillas. Solo se hará referencia al funcionamiento de StrutsCX para comentan los problemas surgidos durante la implementación y como resolverlos.

[5] Según se define en J2EE

6. Guía para el Desarrollador

El objetivo de este punto es servir de punto de inicio y guía a los programadores y diseñadores que continúen la construcción de otras funcionalidades de la Intranet.

Las explicaciones y comentarios de este apartado son técnicas y como no podría ser de otro modo, están dirigidas a:

- Programadores familiarizados con conceptos SAX / DOM, parseadores de XML como Xerces, Xalan..., la construcción de objetos DOM (JDOM, DOM4J...). Para profundizar en estos conceptos recomendamos el libro de aprendizaje **[XSLT01]**.
- Diseñadores con un nivel medio / avanzado del funcionamiento XSL, motores de transformación, árboles DOM, XPath y cierta experiencia con la sintaxis de XSLT. Para profundizar en estos conceptos recomendamos el libro de referencia **[XSLT02].**

Estos conocimientos son necesarios, entre otras cosas porque para terminar de entender lo que aquí se explica y profundizar en la materia es necesario acudir al código fuente de las acciones (*Action.java) y a las plantillas (*.xsl) que esta perfectamente comentado.

6.1 Arquitectura de las plantillas XSL

Las plantillas se pueden entender como *"pequeños programas que reciben documentos XML como parámetros y producen páginas HTML como resultado"*, por lo que teniendo nociones sobre programación, se puede comprender sin problemas, la arquitectura utilizada.

La arquitectura elegida ha sido una libre adaptación de la propuesta en **[XSLT02]**, no obstante mantiene sus características:

Esta recomendada para agilizar:

- El mantenimiento de las pantallas.
- El desarrollo de nuevas pantallas.
- Facilitar el cambio de "*look"* de la aplicación.

Sin embargo, aunque se trata de una arquitectura simplificada y fácil de entender, requiere mucho tiempo de implementación y prueba, además se necesita conocimientos avanzados sobre el lenguaje de XSL para llevarla a acabo. Por lo que no resulta recomendable para:

- Proyectos cortos en duración.
- Aplicaciones con pocas pantallas.
- Para desarrolladores con poca experiencia.

La arquitectura está basada en cuatro tipos de *hojas de estilo* XSL

1) Utilidades

Cada una de estas hojas de estilo recogidas en la carpeta Utils, proveen de utilidades al resto de plantillas. Ejemplos de estas utilidades son:

- Facilitar el acceso a constantes.
- Plantillas para manejar texto, rutinas de búsqueda y sustitución...
- Interprete de tags HTML...

2) Layouts

Estas hojas definen el diseño del documento final, con ellas se intenta reutilizar código y conseguir una apariencia uniforme entre todas las páginas.

Un layout define la estructura de una página o parte de ella.: Una cabecera, dos columnas y un pie de página.

Basándose todas las páginas en los mismo layout se consigue una apariencia uniforme en toda la aplicación.

3) Componentes

Los componentes son pequeñas plantillas que se utilizan en varias páginas. Se agrupan en hojas por funcionalidades

Permiten la reutilización de código con las ventajas que conlleva: disminución del número de errores, del tiempo en desarrollo y del tiempo en detectar y corregir errores.

4) Representación de Contenidos

Estás hojas son las que definen el formato de cada página en particular, suele haber una por cada página y se suele utilizar el mismo nombre que la acción que la genera.

Hay que distinguir los **layout** que imponen una estructura, de los **componentes** que implementan una funcionalidad. Es relativamente frecuente, implementar inicialmente un contenido y después encapsularlo como un componente si se repite la funcionalidad en otras páginas, sin embargo, un layout debe ser implementado desde el principio, antes que los contenidos.

Antes de continuar con la estructura, hay que decir que todas los ficheros XSL, ya que son considerados recursos estáticos, se encuentran en la carpeta COLDY/**web**/xsl, de esta forma podrían servirlos por un servidor web como Apache.

No obstante si se quieren proteger estos recursos estáticos, al igual que los .class, habría que moverlos a la carpeta WEB-INF. Teniendo en cuenta que la implementación de los XSL esta especialmente ligada a la estructura de directorios[6] de los recursos estáticos, por lo tanto, si se quieren mover, es necesario mover la carpeta *web* al completo.

[6] La estrecha relación entre la implementación de XSL y la estructura de directorios definida por la arquitectura esta comentada en la sección de problemas.

6.2 Estructura y Diseño

A continuación se expone la estructura y diseño de las plantillas más relevantes, así como su utilización por parte de otras plantillas, si es preciso.

6.3 Utilidades

En el momento de escribir este documento hay cuatro plantillas de utilidades

- **dateUtils**: Conjunto de plantillas para formatear y presentar fechas correctamente y no depender del formato en el que JAVA las muestra. No obstante, se recomienda utilizar siempre que sea posible, la clase *Locale* de JAVA en lugar de estas utilidades.
- **stringUtils**: Estas plantillas se utilizan como funciones para buscar y sustituir caracteres dentro de un bloque de texto.
- **HTMLHelper**: Si a la hora de generar los datos, se quiere precisar que formato hay que darles se puede incluir en el XML de datos etiquetas propias de HTML, entonces se utilizan estas plantillas para interpretarlo. Esta práctica no es recomendable.
- **Variables**: Reúne un conjunto de variables muy utilizadas en el resto de plantillas.

Para utilizar estas plantillas es necesario incluirlas donde se vayan a utilizar. Como dice la especificación XSL, los motores de transformación solo trabajan con un fuente XML y una plantilla XSL, cuando una de ellas tiene *includes* el motor de transformación añade la plantilla incluida en la original, cada motor resuelve a su manera las posibles incompatibilidades. Debido a esto sólo es recomendable incluir las utilidades y cualquier otra hoja en una de las hojas de estilo aplicadas, esto se suele hacer en la primera.

Estas plantillas deben ser llamadas explícitamente (no por concordancia de patrones) y pasarle los parámetros necesarios que cada una necesite.

A continuación se explica más detalladamente la plantilla variables como ejemplo de utilidad.

Variables.xsl

Las variables de XSL se comportan como las constantes de cualquier otro lenguaje, esto es porque una vez tengan asignado un valor no se puede cambiar. Las variables tampoco pueden eliminarse y se sobre entiende que son de tipo texto.

En esta plantilla se declaran e inicializan muchas variables (constantes), las variables están agrupadas por funcionalidad:

- Path: rutas a las imágenes mas utilizadas.
- Entidades importantes, como el usuario que esta autenticado, el idioma...
- Caracteres especiales, como el espacio, copyright...

6.4 Layouts

La función de cualquier layout es definir el formato de la página, por ello son los encargados de escribir los tag principales del documento final:

<html> <head>…</head> <body>…</body> <html>

Los layout deben ser invocados al tratar la primera etiqueta del documento fuentes, por lo tanto, todos concuerdan con el patrón "/" (nodo raíz).

Por el momento sólo se ha implementado un layout

PrincipalLayout.xsl

Este layout define el formato de todas la páginas de la aplicación, dividiendo la pantalla en tres zonas:

- Cabecera
- Contenido principal
- Pie

Este patrón utiliza algunas variables declaradas y calculadas en la hoja principal, podrían pasarse como parámetro pero esta forma de uso es más optima y son valores obligatorios como el título de la ventana...

Se recomienda utilizar estos nombre, si se cambian, habría que actualizar los layouts:

- **pageTitle**: Para representar el *caption* de la ventana.

6.5 Components

Por el momento, solo se han necesitado dos tipos de componentes:

- Navigation: La barra de navegación que informa de la pantalla actual y cuales han sido las anteriores, se ha aislado como componente ya que esta presente en casi todas las pantallas y con la misma lógica.
- Tales: Es un conjunto de leyendas y textos que aparecen a menudo como el usuario actual, la fecha, copyright...

Los componentes pueden recibir ciertos parámetros que configuren su comportamiento.

Hay que distinguir claramente la diferencia entre:

- Componente y Utilidad:

Aunque ambos tienen argumentos, las utilidades se caracterizan por tratar el valor de los argumento, modificarlos... y devolver nuevos valores a partir de ellos. Es decir, se comportan como funciones, mientras que los componentes son procedimientos, hacen algo que se debería hacer en el programa principal pero es aislado por que se puede reutilizar.

- Componentes y Layout:

Ambos contribuyen a formatear la página destino, sin embargo, los componentes no estructuran la página sólo la decoran y los layout hacen lo contrario, de hecho un layout no debe presentar información.

Navigation.xsl

La acción que genera el fuente XML debería incluir la etiqueta "*navigationpages*" con las siguientes características:

- Sólo debe existir una etiqueta navigationpages y debe ser un nodo de *root/data.*
- Habrá una etiqueta "*page*" por cada página visitada.
- Estarán colocadas según el orden de visita.
- La etiqueta *page* consta de un atributo "***name***" que es el nombre de la página y una etiqueta "***link***" con el path a la página.

Este componente recorre cada *page* con un bucle *for-each*, creando la barra de navegación.

Evidentemente, el uso de esta plantilla requiere que la acción genere el fuente XML adecuado, para tratar de automatizar esta tarea, en la acción base se guardan ordenadamente en la sesión de usuario todas las páginas visitadas, creándose de esta forma un objeto que después utilizará el Document Builder para construir el xml fuente de esta plantilla.

6.6 Contenido

En principio se creará una hoja de estilos de contenidos por cada pantalla.

La estructura de estas páginas es común:

1.- Incluyen las utilidades necesarias, en especial las variables.

2.- Crean las variables necesarias para el layout que se quiere utilizar.

3.- Se invoca el layout deseado.

4.- En la plantilla "***content***" invocada por el layouyt, se genera el contenido principal de la página destino con ayuda de los componentes y utilidades.

Estás hojas como el resto tienen acceso a todo el documento fuente, sin embargo se centran sobre todo en el contenido de *data*, ya que su objetivo es formatear dicha información.

6.7 Notas Técnicas

1. Alojamiento de la Información

Además del contenido que va a modelar, una plantilla necesita ciertos datos para generar el documento final, por ejemplo:

- El path de los recursos estáticos imágenes, css...
- Mensajes para copyright y otras leyendas
- Versión, idioma...
- Constantes

Las plantillas XSL pueden obtener información de tres sitios.

- Del documento XML al que son aplicadas. Además, de los datos a presentar puede contener información "extra" de la session, request... como se ha visto el anteriormente.
- De otras plantillas XSL, utilizando las instrucciones *include* o *import*.
- De si mismas, aunque no resulta escalable guardar datos independientes del modelado en cada plantilla.

Es importante saber donde se debe alojar y extraer cada tipo de información. Analizaremos esto con un ejemplo:

Ejemplo: Información de la página principal (HOME)

Se necesita mostrar la siguiente información en la HOME:

- Nombre del usuario logado.
- Hora actual del sistema.
- Idioma utilizado.
- Fecha de la última actualización.
- Versión de la aplicación.
- Mail del administrador o webmaster.
- Tabs de Navegación
- Copyright
- Logotipos
- Mensaje de Saludo en función del idioma.
- Menú general con varios links.
- Mensajes como navegador recomendado...

Además se necesitan los siguientes recursos:

- css
- texturas e imágenes

Para saber donde alojar esta información es necesario clasificarla de alguna forma, dos buenos criterios para son el **origen** y **destino(s)** de la información, distinguir entre información ***estática*** o ***dinámica*** y si es propia de la página o común a otras.

Atendiendo a estos criterios podemos hacer la siguiente clasificación:

Múltiples destinos: Común al resto de páginas

- Origen Dinámico
    - Hora actual del sistema.
    - Nombre del usuario logado.
    - Idioma utilizado.
    - Tabs de Navegación
- Origen Estático
    - Fecha de la última actualización.
    - Versión de la aplicación.
    - Mail del administrador o webmaster.
    - Copyright
    - Logotipos
    - Recursos: css, imágenes...

Un único destino: Propia de la página principal

- Origen Dinámico
    - Menú general con varios links.
- Origen Estático
    - Mensaje de Saludo en función del idioma.
    - Mensajes como navegador recomendado...

Una vez clasificada la información estudiaremos donde alojar cada tipo.

Información común y dinámica

Distinguimos entre dos tipos más:

- Información común a todos los usuarios, como por ejemplo: Hora actual del sistema.

  Esta información debería residir en el contexto de la aplicación, sin embargo el DocumentBuilder estándar de StrutsCX no recoge esta información, podemos alojarla en la session y la recuperarla del tag "session" o extender la funcionalidad de StrutsCXDocumentBuilder.
- Información común a la sesión de usuario, como por ejemplo: Nombre del usuario logado, Idioma utilizado...

  Esta claro que esta información se obtiene de la session. Se puede tener una acción básica que se encargue de introducir esta información en la session en forma de atributos o parámetros.

En conclusión cualquier información común y dinámica se introduce en la session, a menos que otras restricciones nos obliguen a implementar un DocumentBuilder personalizado. No obstante, para facilitar la inclusión de datos comunes en la session se ha desarrollado una clase abstracta *edu.ucm.sip.inet.web.actions.GeneralAction* ver JavaDoc.

Información común y estática

Estos datos solo cambia en momento puntuales como en el cambio de versión, actualización de contenidos, es decir, prácticamente es información constante por eso se puede tener almacenada en ficheros, base de datos... No obstante acceder a recursos externos puede ralentizar el proceso de transformación, por lo tanto, es mejor optar por incluirla en una plantilla XSL.

En el caso de depender del idioma u otro parámetro como puede ser el texto del copyright se puede optar por utilizar un recurso externo o incluir lógica en las plantillas XSL.

Información particular y dinámica

Esta información requiere ser calculada en el momento, además suele depender del estado actual y anteriores de la sesión del usuario, por lo tanto, requiere de cierta lógica que según sea en mayor o menor medida complicada, será implementada por el controlador en la propia acción o como parte del negocio en la capa del modelo.

Debido a que el programador interviene de un modo muy activo en su generación el tag ideal es *data*.

En el ejemplo expuesto, se ha decido crear un objeto XML en la acción con la siguiente estructura:

```
<navigationpages>
  <page name="Init">
    <link>../index.html</link>
  </page>
  <page name="HOME">
    <link>welcome.do</link>
  </page>
</navigationpages>
```

En concreto, este ejemplo, presenta una dificultad añadida. Para formar los links necesitamos información dinámica que veremos después como obtenerla.

Información particular y estática

Hay dos formas fáciles de generar esta información:

1.- Generando el mensaje desde código, leyéndolo desde un XML externo en función del idioma.

2.- Introduciendo el mensaje en el fichero de recursos.

La primera opción ofrece la ventaja de no recargar un XML común para todas las páginas, por el contrario ofrece más complejidad a la hora de implementar la acción.

En cuanto a la recuperación de información es igual:

1.- <xsl:value-of select=’ root/data/welcome_msg/’/>

2.- <xsl:value-of select=’ resourcesproperties/welcome/welcome_msg />

Qué método utilizar dependerá de la longitud del mensaje y de la posibilidad de aparecer en otras páginas.

2. Formación de Links

Un servidor de aplicaciones, establece una ruta base por cada aplicación que levanta, formando a partir de ella el árbol de directorios correspondiente, en concreto tal y como esta montado el entorno de producción con Jboss tenemos:

http://servidor.dominio:puerto/appname

/*.do → Mapeos a las acciones

/web/i → imágenes

/web/css → Hojas de estilo en cascada

/web/xsl → Hojas de estilo

Lo deseable es formar links relativos independientes del nombre del servidor, del nombre de la aplicación...

Para poder formar links relativos, una cosa debe estar clara, donde estamos en cada momento, es decir, si ponemos "." a qué directorio estamos accediendo. Si no modificamos la configuración de Jboss podemos suponer que en una hoja XSL acceder, escribir "./" en un path nos posicionaria en "http://servidor.dominio:puerto/appname"

Entre los links que necesitamos destacan dos tipos:

- Links a imágenes: El path "*./web/i/...*", esta disponemos en la variable "***img.dir***" declarada en *variables.xsl*. Por lo tanto, formar un link a una imagen sería algo como: **, aunque las imágenes más frecuentes tienen sus propias variables.: **.

- Links a acciones: El path "*appname*", esta disponible en la variable "***project***", esta información se extrae de la sesión en el atributo "*projectName*" Por lo tantom formar el link se reduce a "*action="{$project}nombreaccion.do*".

También podemos utilizar la variable "***jsessionid***" para formar link a otras acciones sin perder la sesión de usuario.

3. Tratamiento de errores

El tratamiento de errores es muy importante para la depuración durante el desarrollo, además influye notablemente en la opinión del usuario sobre la fiabilidad de la aplicación.

La solución adoptada sobre este tema se explica en un documento aparte ya que afecta a todas las capas de la aplicación.

Durante el desarrollo de la capa de presentación hemos tenido que afrontar numerosos problemas, de los cuales hemos resuelto muchos y otro no, a continuación se explican los más importantes:

4. Problemas Resueltos

Los Acentos

No nos ha sido ajena la confusión en el mundo de intenet entre "*charset*" y "*encoding*". Explicaremos brevemente estos dos conceptos, antes de explicar el problema:

- **Charset**: Un charset como su nombre indica es un conjunto de caracteres, por ejemplo: el abecedario español de 1803: "a, b, c, d, e, f, g, h, i, j, k, l, m, n, ñ, o, p, q, r, s, t, u, v, w, x, y, z" sin "ch" y "ll", aunque se necesitan más caracteres para escribir en español: signos de puntuación, tildes...

  Diversas organizaciones internacionales, como la w3c, han estandarizado muchos charset, entre ellos:

  - ASCII: Caracteres ingleses normales.
  - Latin1: Caracteres normales en Europa Oeste.
  - Latin2: Caracteres normales en la Europa del Este.
  - Unicode: Todos (o casi todos) los caracteres conocidos en el mundo.

- **Encoding**: Un encoding dicta las reglas para codificar un carácter. Por ejemplo: Codificar de acuerdo al orden binario cada carácter formando bloques de 7 bits.

Al igual que con los charset, organizaciones internacionales han ideado varios encoding, los más destacados actualmente son:

- UTF-8
- UTF-16
- ISO-8859-X (X=1, 2..9)

El problema es que los encoding se han hecho de acuerdo a un charset determinado, por ejemplo: el ISO-8859-1 dicta las reglas para codificar el Latin1, el Unicode puede cópdificarse con UTF-8, UTF-16.. de aquí la confusión entre charset y encondig.

En qué nos afecta a la aplicación

Todos los documentos XML (incluidos XHTML, XSL...) deben tener una cabecera[7] que indique el charset con el que se ha escrito y el encoding para descifrarlo, puesto que los encoding estandarizados tienen un charset implícito, este no hay que explicitarlo

Entonces para poder escribir acentos en un XSL es necesario que se especifique un encoding adecuado como el **ISO-8859-1**. Utilizamos UTF-8 en los XML de mensajes para facilitar la i18n con cualquier lenguaje, en estos casos, tendremos que escapar convenientemente los acentos.

Si no respetamos esta norma, se producirán errores bastante difíciles de depurar durante los procesos de transformación.

[7] <?xml version="1.0" encoding="ISO-8859-1"?>

Internacionalización (i18n)

La forma de soporta la i18n por StrutsCX implica mantener en memoria y en todos los documentos a descargar el contenido de todos los mensajes internacionalizados, se utilicen o no en ese momento.

Nosotros hemos optado por esta solución nada más que para los mensajes comunes o muy frecuentes como los textos de copyright, fechas...

Los textos particulares de cada página, se tienen en un xml con el nombre de la acción que la genera y el sufijo correspondiente al idioma. Estos ficheros tienen siempre la misma estructura:

```
<messages>
    <caption>Welcome to</caption>
    <title>DSIP Intranet</title>
     <subtitles>
     <subtitle name="Login">
          <link>logon.do</link>
       </subtitle>
    </subtitles>
     <..../>
  </messages>

<messages>
    <caption>Bienvenido a</caption>
    <title> Intranet DSIP</title>
     <subtitles>
     <subtitle name="Entrar">
          <link>logon.do</link>
       </subtitle>
    </subtitles>
     <..../>
</messages>
```

Lo primero que hacen todas las acciones es cargar el xml con los literales correspondientes e incluirlos en el XML fuente sobre el que aplicará un xsl.

Conexión a internet

Ciertas clases utilizadas validad los xml en caso de tener una DTD asociada, esto implica acceder a la dtd y parsearla, esto no solo supone una perdida de tiempo sino que puede provocar errores en caso de no tener acceso a la DTD (normalmente en Internet).

Por lo que se han eliminado las referencias a DTD en muchos XMLs.

5. Problemas sin resolver

Validadores

En algunas ocasiones se produce una ReflectionException.

No se ha encontrado una secuencia de reproducción concreta ni tampoco en todos los servidores, aunque solo se ha producido si se utiliza la validación por mascara "<*mask*>"

Se piensa que se debe alguna configuración concreta del servidor de aplicaciones.

Normalmente reiniciando los servicios de JBOSS se corrige.

Parseo en cliente y servidor a la vez

Ha sido imposible encontrar una ruta relativa que permita aplicar los dos tipos de parseo sobre un mismo XSL.

Una de las alternativas estudiadas es incluir lógica extra en los XSL para que se apliquen uno u otro paz según el parseo utilizado, puesto que esta alternativa complica la implementación de los XSL y no tiene especial utilidad se a desestimado.

Como workaround se han mantenido dos XSL uno en cliente (web-client) y otro en server (web/xsl).

6.8 Recomendaciones

De la experiencia obtenida al desarrollar las páginas de la aplicación, hemos concluido una serie de recomendaciones personales a la hora de implementar nuevas pantallas o modificar las actuales.

La Regla de Oro: "Desacoplamiento"

La *regla de oro* de la programación orientada a objetos es garantizar un mínimo acoplamiento entre módulos, para ello se disponen de herramientas como el polimorfismo y la herencia.

Pues bien, durante el análisis, diseño e implementación tanto de las acciones (java), como de las hojas de estilo (xslt), siempre hay que tener en cuenta el acoplamiento entre las distintas capas de la aplicación y utilizar las herramientas disponible para poder garantizar en todo momento por encima de todas las cosas el desacoplamiento entre ellas.

Siempre es inevitable un mínimo acoplamiento, en este caso la cadena esta clara:

- Si cambian los **datos**, cambian los **XML** generados por las acciones.
- Si cambian los **XML**, las **plantillas** deben actualizarse con los nuevos path.
- Si cambian las **plantillas** al incluir o eliminar datos, el **diseño** cambia automáticamente.

Evidentemente, el diseño debe depender de los datos lo menos posible.

Hay que plantearse estas preguntas:

¿Cómo de flexible es el modelo? ¿qué datos afecta realmente a la estructura del XML?

Una estructura rígida del XML facilita la implementación de los xsl, pero aumenta la probabilidad de que un cambio en el modelo afecte a toda la aplicación.

¿qué dependencia hay entre una acción y el resto?¿es realmente configurable?.

La lógica de una acción debería esta aislada del resto de acciones para poder separar realmente una página del resto, en el caso de que existan dependencias como por ejemplo: *login*, *logon* y *logoff* garantizar que dicha funcionalidad no afecta al resto y si afecta externalizarlo de alguna forma, en este caso se necesita un String que identifique un atributo (currentUser) en la session, este nombre debería estar en un XML de configuración, en el árbol JNDI o al menos como una constante.

¿Cómo de flexible es el diseño? ¿cuántas cosas puedo poner/quitar y mantener la misma apariencia de la página?

Un diseño simple y con "*huecos*" nos abre la posibilidad a incluir nuevos contenidos y facilita la adaptación de las páginas a varios navegadores.

Recomendaciones durante el Desarrollo

En nuestra opinión la principal ventaja que aporta la utilización de XML/XSL es la separación de roles, el resto de ventajas se ven compensadas con las desventajas y además existe otras tecnologías con las mismas prestaciones y no tan engorrosas como XSL.

Sugerimos dos recomendaciones sobre este punto:

- Se establezca un responsable del modelo y otro para la vista. Si es la misma persona, debe saber colocarse en cada situación y afrontar los problemas independientemente.
- Establecer un acuerdo inicial sobre la estructura de los datos y fijar así un XML de plantillas (los decoradores vistos en la introducción). A partir de este punto el responsable del modelo debe asegurar que los datos se ofrecen de esta manera y el de la vista implementar las plantillas de acuerdo a dicho formato.

Recomendaciones el Diseño

- El diseño debe ser lo más simple posible y uniforme, en general, respetar las normas básicas respecto al diseño de páginas web (colores, estructura...).
- Tener en cuenta la resolución de los dispositivos que visualizan las pantallas.
- Tener en cuenta el software (navegadores) que presentaran las pantallas.
- Utilizar los layouts existentes o crear unos nuevos, no estructurar una página directamente desde la hoja de estilos de contenidos.

Recomendaciones en la Implementación

- Utilizar buenas herramientas para manejar los datos y trabajar con los XSL.
- Los comentarios no son gratis, también se descargan lo que conlleva tiempo, por otro lado son imprescindibles, por la forma de funcionar los motores de transformación, es imposible seguir hojas de estilo sin comentarios en cuanto tenga más de tres includes, así que hay que hacer un buen uso de los comentarios pero no abusar de ellos.
- El lenguaje XSL soporta instrucciones de control (if, while...) sin embargo, debe utilizarse para facilitar la presentación y no implementar en una hoja xsl, lógica perteneciente al modelo, por motivos de eficiencia y claridad.
- Aislar en componentes la lógica repetitiva.
- Ayudarse de las utilidades para manejar los datos.
- Implementar más layouts principales.
- Utilizar layouts de segundo nivel, que modelen sólo ciertas partes de la página, si son necesarios.

Bibliografía

Patrones

[GOF94]
Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (The Gang of Four). *Design Patterns*. Addison Wesley Professional Computing Series, 1994.
[PATTSUN]
http://java.sun.com/blueprints/corej2eepatterns/index.html
[PATTPC]
http://www.programacion.net/tutorial/patrones/

Servlets y JSP

[SUNWEB1]
Desarrollo de Componentes Web con la Tecnología JavaTM (SL-314)
Manual del Alumno (Sun Microsystems, Inc)
[JSPPC01]
http://www.programacion.net/tutorial/servlets_jsp/

XML

[XMLPC01]
http://www.programacion.com/tutorial/apuntesxml/
[XMLPC02]
http://www.programacion.net/tutorial/javaxml/
[XMLPC03]
http://www.programacion.net/tutorial/jspyxml/
[XMLPC04]
http://www.programacion.net/tutorial/xhtml/

XSL

[XSLPC01]
http://www.programacion.net/articulo/xslt/
[XSLT01]
Michael, Kay.2001. *XSLT Programer's Reference*, Wrox Press 2. Wiley, DA (2000).
[XSLT02]
Eric M. Burke. *Java and XSLT* (Oreilly Java)

Spring

[SPR01]
http://www.springframework.org
[SPR02]
http://www.theserverside.com/articles/article.tss?l=SpringFramework

Struts

[JAKA03]
http://jakarta.apache.org/
[STRUTS]
http://jakarta.apache.org/struts/
[STPC1]
http://www.programacion.net/tutorial/joa_struts/
[STPC2]
http://www.programacion.net/tutorial/struts/
[STPC3]
http://www.programacion.net/articulo/tips_struts/
http://www.theserverside.com/articles/article.tss?l=Struts1_1

Struts + XSL

[MERC01]
Julien Mercay and Gilbert Bouzeid.
***Boost Struts With XSLT and XML*. Java World, Febrero 2002**.
[ORB01]
http://www.orbeon.com/oxf/doc/model2x-model2x

[STXX1]
http://stxx.sourceforge.net/
http://stxx.sourceforge.net/xmlforms/configuration.html
[STXX2]
http://www.javaworld.com/javaworld/jw-02-2002/jw-0201-strutsxslt.html
[STXX3]
http://www-306.ibm.com/software/globalization/topics/struts/example.jsp

[STCX1]
http://it.cappuccinonet.com/strutscx/index.php
[STCX2]
http://www.devx.com/Java/Article/11381/0/page/1
[STCX3]
http://news.gmane.org/gmane.comp.java.strutscx.user/cutoff=191

Validator

[JAKVAL01]
http://jakarta.apache.org/struts/api/org/apache/struts/validator/package-summary.html
[JAKVAL02]
http://wiki.apache.org/geronimo/Architecture/Validator
[WINTED01]
Co-authored by David Winterfeldt ant Ted Husted.
Struts in Action, Chap. Validator User Input.
http://java.sun.com/developer/Books/javaprogramming/struts/struts_chptr_12.pdf

CONTROL DE VERSIONES

Este documento no esta cerrado y se irá ampliando según sea necesario hasta finalizar el proyecto.

1.0→ Conceptos

1.1→ Estudio de las soluciones del mercado (Struts y STxx)

1.2→ Presentación de StrutsCX

2.0→ Arquitectura, diseño e implementación con StrutsCX

2.1→ Implementación, resumen de los problemas encontrados.

2.1a→ Revisión y maquetado del documento.

2.2→ Problemas no resueltos.

APÉNDICE 1

1. Ejemplo de XML

El siguiente xml se ha obtenido aplicando el parámetro xmldebug=true en la página de bienvenida, para más información FAQ StrutsCX[8].

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<root>
  <!-- ********** DATA ********** -->
  <data>
    <messages>
      <titlewin>Welcome to</titlewin>
      <nolog>You must identified before continue</nolog>
      <title>DSIP Intranet</title>
      <subtitles>
        <subtitle name="Login">
            <link>logon.do</link>
        </subtitle>
      </subtitles>
    </messages>
  </data>
  <!-- ********** RESOURCES-PROPERTIES ********** -->
  <resourcesproperties lang="en">
    <!-- Mensajes Generales -->
      <lastupdate>May 1, 2004</lastupdate>
      <lastupdatetext>Last update</lastupdatetext>
      <ucm>Complutense University of Madrid</ucm>
      <appname>Intranet of Computed and Programming Systems
        Departament</appname>
      <logoff>Log off</logoff>
      <textadmin>Webmaster</textadmin>
      <textquestionadmin>DSIP web question</textquestionadmin>
      <debug>Show XML source</debug>
  </resourcesproperties>
```

[8] En el entorno de producción esta opción esta desactivada, hay que activarla en el strutscx-config.xml

```
<!-- ********** ACTIONFORM **********  -->
<actionform>
  - <LogonForm>
     <class>class
        edu.ucm.sip.inet.web.auth.actions.LogonForm</class>
      <multipartRequestHandler />
      <page>0</page>
      <password />
      <resultValueMap />
    <ActionServletWrapper>
      <class>class
        org.apache.struts.action.ActionServletWrapper</class>
      </ActionServletWrapper>
      <username />
      <validatorResults />
    </LogonForm>
  </actionform>
<!-- ********** ACTIONERRORS ********** -->
<errors>
    <password />
  </errors>
<!-- ********* REQUEST ********** -->
<request>
  <!-- Attributes  -->
  <xsl_key>login</xsl_key>
  <org.apache.struts.action.MESSAGE>org.apache.struts.util.Propert
      yMessageResources@ca4aae</org.apache.struts.action.MESSA
      GE>
    <xsl_clientside>false</xsl_clientside>
    <encoding_key>ISO-8859-1</encoding_key>
  <!-- Parameters  -->
    <debugxml>true</debugxml>
    <password />
    <username>113</username>
  </request>
```

```
<!-- ********** SESSION ********** -->
<session>
    <jsessionid>C16577B491E5AA9CD0FE3E0D7...</jsessionid>
  <com.cappuccinonet.strutscx.domainparams_flag>
    true</com.cappuccinonet.strutscx.domainparams_flag>
  <date_time_today>
      <string>Sun May 23 19:29:54 CEST 2004</string>
      <era>121</era>
      <year>2004</year>
      <month>5</month>
      <week_of_year />
      <week_of_month>3</week_of_month>
      <date>23</date>
      <day_of_month>23</day_of_month>
      <day_of_year>144</day_of_year>
      <day_of_week>1</day_of_week>
      <day_of_week_in_month>4</day_of_week_in_month>
      <am_pm>1</am_pm>
      <hour>7</hour>
      <hour_of_day>19</hour_of_day>
      <minute>29</minute>
      <second>54</second>
      <millisecond>971</millisecond>
      <zone_offset>1</zone_offset>
      <dst_offset>1</dst_offset>
    </date_time_today>
    <domain>localhost</domain>
    <rootpath>...\jboss-
      3.2.3\server\...\tmp\deploy\tmp61892coldy.ear-
      contents\coldy.war\</rootpath>
    <strutscx_version>v0.9.5</strutscx_version>
    <session_flag>true</session_flag>
    <httpDomain>http://localhost:8080</httpDomain>
    <port>8080</port>
    <projectName>/coldy</projectName>
    <locale>en_</locale>
    <language>en</language>
    <country />
  </session>
</root>
```

CAPA DE MODELO

Intranet para un Dpto. Universitario

3 de Julio de 2004, v1.0

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

Documentos Involucrados

- ANÁLISIS Y DISEÑO DEL MODELO DE DATOS
- ENTERPRISE JAVA BEANS
- ACCESO CONCURRENTE Y BASE DE DATOS
- OPTIMIZACIÓN DE BASES DE DATOS ORACLE

INTRODUCCIÓN A LA CAPA DE MODELO

A continuación se han agrupado una serie de documentos que pretenden dar toda la información necesaria para comprender a incluso poder continuar o mejorar la capa de Modelo del Proyecto.

La capa de Modelo engloba los Datos Persistentes de la aplicación, el acceso a ellos y su gestión y ciclo de vida. Además se ha estimado conveniente añadir apéndices sobre tecnologías concretas que han sido seleccionadas para dar soporte al Modelo de Datos, de modo que la estructura de la aplicación pueda ser comprendida y gestionada sin problemas una vez se haya terminado la lectura de los documentos englobados en esta sección.

Los documentos aquí agrupados responden, entre otras, a las siguientes preguntas fundamentales:

- ¿Qué datos necesita almacenar la aplicación?
- ¿Cómo van a estar organizados y cómo se van a relacionar entre sí?
- ¿Cómo se va a acceder a esos datos desde la aplicación?
- ¿Qué fabricante de SGBD ofrece el soporte lógico más recomendado para nuestro proyecto?
- ¿Cómo será gestionada y salvaguardada la información por el SGBD seleccionado?
- ¿Cómo puede ampliarse y mejorarse el rendimiento del acceso a datos a medida que la aplicación vaya creciendo?

La forma en que ha sido divididos atiende a razones prácticas. Los documentos siguen un orden lógico que ha sido seguido paso por paso en la confección del proyecto.

En primer lugar, en el documento Análisis y Diseño del Modelo de Datos se da toda la información relativa a la estructura de los datos, repasando los conceptos teóricos que se han aplicado en cada y dando explicaciones razonadas a cada elección efectuada. Además se dan las pistas que se han seguido para seleccionar el mejor gestor de Bases de Datos en función a los criterios seguidos en la confección del modelo de datos y también fijando la atención sobre las posibilidades adicionales que ofrece al proyecto.

En segundo lugar se plantea el modo en el que esos datos almacenados en el SGBD van a ser accedidos desde la aplicación web. La tecnología escogida, los Enterprise Java Beans, es presentada inicialmente en su conjunto para después centrar la atención sobre el tipo de EJBs más afín con el objetivo del proyecto.

En tercer lugar, una vez definidos los datos, cómo se almacenan, y cómo se accede a ellos se plantea el problema más habitual presente en toda aplicación web: Concurrencia.

Múltiples usuarios pueden estar accediendo a los datos en el mismo instante de tiempo y dichos datos deben ser salvaguardados de posibles interferencias o corrupciones. El SGBD y los EJBs son los encargados de proporcionar un acceso concurrente libre de estos errores. El modo en que se garantiza esta protección es explicado con numerosos ejemplos y con gran detalle en esta sección.

Por último, llegados a esta fase del diseño cabe pensar que la aplicación puede llegar a crecer con el tiempo y por tanto su volumen de datos pueda llegar a ser mucho mayor que el esperado inicialmente. O también puede ocurrir que los requisitos de la aplicación cambien con el tiempo y la velocidad de obtención de los datos pase a ser un factor crítico mucho tiempo después de que la aplicación este finalizada. En este último apartado se enumeran las más importantes técnicas de las que se dispone en el SGBD Oracle para optimizar al máximo el rendimiento del acceso a los datos, atendiendo a los tres factores susceptibles de mejora: CPU, Memoria y Entrada/Salida.

Está demostrado que ligeras modificaciones en el modo de acceso a ciertos sistemas de almacenamiento en BBDD pueden dar como resultado mejoras drásticas en el rendimiento de las aplicaciones, incuso de varios órdenes de magnitud superiores a las mejoras que puedan ser introducidas en el código de este tipo de aplicaciones, por ello se ha considerado no sólo útil sino imprescindible una amplia referencia a este campo, siempre en constante avance en el mundo de la informática: la optimización.

ANÁLISIS Y DISEÑO DEL MODELO DE DATOS

05 de Mayo de 2004, v2.1

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

1. INTRODUCCIÓN

El objetivo de este documento es exponer los fundamentos sobre los que se base el modelo de datos de la aplicación para facilitar su ampliación, mantenimiento y posibles correcciones.

El documento esta divido en tres partes principales:

1. Análisis: Primer acercamiento y profundización en Oracle.
2. Diseño: Explicación del diseño y definición de las entidades que forman el modelo de datos.
3. Implementación: Conjunto de scripts en SQL utilizados para crear el modelo de datos.

El resto de secciones son nexo y aclaraciones que se han considerado necesarias entre cada sección principal.

2. CONSIDERACIONES PREVIAS

A la hora de diseñar una aplicación que maneja datos como la que aquí se expone, se presentan dos alternativas para conseguir la persistencia de los datos:

- Almacenar los datos del programa en disco mediante la serialización de los TAD´s (Tipos Abstractos de Datos) que maneja el programa.
- Almacenar en una base de datos la información contenida en los TAD's. En la actualidad esta opción va ligada a la utilización de un Sistemas Gestores de Bases de Datos (SGBD) que prestan numerosos servicios como la gestión de usuarios concurrentes, control de transacciones...

Se ha escogido ésta segunda opción por las siguientes ventajas:

1. Escalabilidad de la solución.
2. Sistema probado, más versátil, potente y robusto.
3. Modularidad de la solución.
4. Herramientas y utilidades propias.
5. Ahorro considerable en tiempo, ya que se evita tener que programar ciertos comportamientos que ya proporciona un SGBD.

Una vez tomada ésta primera decisión, hay que decidir un SGBD concreto. Para filtrar la amplia gama disponible en el mercado tanto de pago como de libre distribución, fuimos orientados por expertos y revistas técnicas, reduciendo el estudio a cuatro productos.

- Oracle 9i. El más utilizado en el mundo.
- MySQL 4.0. Muy buena fama en el mundo del software libre.
- PointBase. Integrado con el entorno de desarrollo de SUN.
- MS Access 2000. Muy fácil de usar.

Para decidirse por uno de los cuatro se enumeraron los requisitos que consideramos más importantes y se calificó cada sistema por separado.

Requisito	Oracle	MySQL	PointBase	Access
Integración con JAVA[1].	5[2]	5	5	4
Soportan SQL estándar.	5	5	4	2
Precio	5[3]	5	5	1
Documentación abundante y de calidad	5	4	3	5
Uso en el mercado	5	5	1	4
Experiencia del equipo de trabajo.	5	3	1	5
Total:	**30**	27	19	21

Además **Oracle**:

- Dispone de mecanismos para la gestión de la concurrencia, manejo de transacciones y mantenimiento de la integridad de los datos transparentes para el programador. Solo la versión de pago de MySQL tiene estas características.
- Ofrece una serie de utilidades al Administrador de Bases de Datos o DBA (DBA Studio…) para administración y gestión de las bases de datos.

Por lo tanto, como indica la tabla, finalmente se eligió Oracle.

3. ANÁLISIS DEL MODELO DE DATOS

Una vez elegido Oracle como SGBD hay que tomar ciertas decisiones sobre qué y cómo se usa. Se ha sometido a estudio gran parte del sistema y como resultado del análisis hemos obtenido las siguientes conclusiones:

3.1 Claves Primarias y Secuencias

Se ha decidido utilizar como clave primaria campos que aporten información a la entidad a la que pertenecen, es decir, utilizar campos como el NIF para identificar a personas y no incluir una secuencia o un autoenumerado para identificar los registros de personas. También recomendamos utilizar claves formadas por varios campos antes que utilizar secuencias.

Esta decisión se ha tomado porque manejar secuencias desde código resulta complicado y hace el código totalmente dependiente del SGBD, ya que Oracle utiliza secuencias, Access utiliza un tipo especial "autoenumerado", MySQL usa colecciones...etc.

[1] JAVA es el lenguaje de programación escogido.
[2] Oracle facilita en su web los Drivers necesarios para comunicar el SGBD con procesos Java a través de conectores JDBC.
[3] Oracle permite el uso de su SGBD de modo gratuito siempre y cuando no se use para fines comerciales, además se puede descargar todo el software en su web http://www.oracle.com/ con el simple hecho de registrarse como visitante.

3.2 Integridad Referencial

La integridad referencial es un sistema de reglas que utilizan la mayoría de las bases de datos relacionales para asegurarse que los registros de tablas relacionadas son válidos y que no se borren o cambien datos relacionados de forma accidental produciendo errores de integridad.

Aplicar estas reglas implica añadir restricciones a la inserción, modificación y borrado de datos. Supone que el SGBD compruebe las reglas, por cada orden que inserten o modifiquen datos de dichas tablas, de forma que si la orden cumple las restricciones el sistema asegura que los datos son coherentes y si no las supera el sistema genera un error. Obligan a que para borrar un registro que es clave ajena se elimine del resto de tablas referenciadas, este borrado de todas las tablas se denomina eliminación *en cascada*.

Las restricciones o asociaciones entre las distintas tablas se definen mediante el uso de Claves Extranjeras (FK[4]), que obligan a que si un dato está en un determinada tabla, también exista en la tabla y campo a los que se hacen referencia la Clave Extranjera.

Por ejemplo: Si decimos que el campo que identifica a una persona en la tabla *Jobs* es clave ajena con la clave principal de la tabla *Staff*, implica que cualquier persona existente en *Staff*, también existirá en *Jobs*, lo que significa que una persona de la aplicación (*Staff*) siempre tendrá un trabajo (*Jobs)*.

En la práctica utilizar claves ajenas tiene las siguientes implicaciones:

1. Afecta negativamente a la eficiencia en la creación, modificación y eliminación de datos en las tablas que tengan aplicada alguna FK.
2. Afecta negativamente a la concurrencia de usuarios ya que la comprobación de las FK debe hacerse por cada conexión independiente lo que implica un cuello de botella.
3. Resulta peligroso ya que un borrado no controlado puede acabar con todos los registros asociados.
4. Resulta práctico durante el desarrollo ya que el SGBD detecta los errores por el programador y este se limita a tratarlos una vez que ocurren.

Cada diseñador toma sus propias decisiones sobre utilizar o no claves ajenas según estás implicaciones. En nuestro caso se ha decidido utilizarlas por los siguientes motivos:

1. Ayudan en la definición del modelo de datos. Una restricción de este tipo no solo obliga al SGBD a verificar que se cumple, sino que informa al que estudia el modelo de datos de las relaciones entre los datos, sin necesidad de estudiar el código de la aplicación que lo maneja.
2. Aceleran el desarrollo, a pesar del riego de eliminar datos importantes (riesgo poco importante en entornos de desarrollo), además ayudan al programador a detectar fallos en el diseño e implementación de la aplicación.
3. Puesto que por un lado estamos trabajando sobre una aplicación abierta y por otro lado se debe avanzar vertical y rápidamente, pero a la vez profundizando en el análisis y diseño, los dos primeros puntos son muy importantes.

[4] FK acrónimo del ingles “Foreign Key”

4. Para mejorar la eficiencia y evitar riesgos en un entorno de producción, las reglas se han generado con scripts independientes de forma que puedan activarse y desactivarse a voluntad del administrador.

3.3 Triggers

Los triggers (disparadores) son una utilidad que aporta Oracle y permiten realizar alguna acción ante determinados eventos, por ejemplo:

Cuando se elimine un usuario, insertarlo en la el histórico de usuarios.

Esta utilidad facilita en gran medida el desarrollo ya que se libera al programador de la responsabilidad de actualizar tablas asociadas, con el consiguiente ahorro de código y las pruebas y errores que deriva.

No obstante, se ha decido no utilizarlos por dos motivos principales:

- El desarrollo se hace dependiente de tener una documentación actualizada constantemente o se pierde el control de lo que se hace. Se da el caso, de que un programador inserta un registro en base de datos y de repente sin saber porque se insertan cinco campos más en otras tantas tablas.
- Dificultan la migración a otro gestor de base de datos.

3.4 Comentarios

Desde el comienzo del desarrollo se ha optado por realizar la documentación durante el desarrollo y mantener una aplicación autoexplicativa, en lugar de mantener un gran volumen de documentación.

Al igual que se ha comentado el código fuente (mediante javadoc), se utiliza el metadato *COMMENT* de Oracle para documentar el modelo de datos. Para ver estos comentarios se puede utilizar herramientas como TOAD o ejecutar consultas como la siguiente:

```
SELECT TABLE_NAME, COLUMN_NAME, COMMENTS FROM ALL_COL_COMMENTS
WHERE     COMMENTS IS NOT NULL AND OWNER='WEBUSER'
          AND TABLE_NAME ='NameTable'.
```

Esta consulta devuelve los comentarios de los campos que forman la tabla indicada por *NameTable*, filtrando los campos que no tienen comentarios o son propios de Oracle.

Esto no dificulta la migración a otro SGBD ya que los metadatos no afectan a los datos, además existen utilidades para transformar esta documentación a formatos como Word y otros.

3.5 Índices y otros

No esta previsto realizar búsquedas exhaustivas por lo que no resulta optimo indexar las tablas, ni siquiera las más grandes.

Existen otras utilidades y herramientas que ofrece Oracle como índices textuales, duplicidad de tablas, backups... que no hemos considerado útiles o adecuadas su utilización, en cualquier caso, no se ha tomado ninguna decisión que impida su utilización a posteriori, por parte del administrador de la aplicación o el DBA[5].

[5] DBA: Acrónimo del ingles Data Base Administrator.

3.6 Histórico

Se entienden por históricos aquellos datos que en algún momento formaron parte de la aplicación aunque actualmente no están activos o fueron eliminados, también se considera información histórica, aquella información que recoge los cambios y el tipo de cambio sobre un registro.

Se consideró útil mantener un histórico de ciertos datos, como pueden ser el personal, salas, material... para contemplar búsquedas sobre ex-empleados, estadísticas de la ocupación de salas o poder averiguar que ha pasado con algún equipo y quien es el responsable. Es decir, el histórico aporta seguridad, facilita la implementación de nuevas funcionalidades y hace posible la recuperación de datos... por estos motivos se le ha dado la suficiente importancia como para analizarlo en profundidad.

Sobre este tema se han considerado muchas alternativas, ya que desde el principio se le ha dado mucha importancia, a continuación solo se explica la opción adoptada.

Se ha procurado tener un histórico independiente de los datos que se quieran almacenar.

Además tener la posibilidad de activarlo o no, al igual que se hace con las Foreing Keys y por motivos similares. Esto último impide recoger la información histórica en una tabla.

Como se ha comentado interesa recoger no solo la información histórica, sino el tipo de acción que la ha convertido en histórica.

Teniendo estas tres condiciones se ha optado por el siguiente diseño:

1.- Una tabla a titulo informativo donde están recogidos los posibles tipos de cambios.

- La tabla es: ***HIST_ACTION_TYPE***.

```
CREATE TABLE HIST_ACTION_TYPE (
       ID_ACTION_TYPE  VARCHAR(1) UNIQUE NOT NULL,
       ACTION_TYPE_DESC VARCHAR2(15) NOT NULL
     );
```

- Y los posibles tipos de cambio son los siguientes: Alta, Baja o Modificación.

```
INSERT INTO HIST_ACTION_TYPE VALUES ('A','ALTA');
INSERT INTO HIST_ACTION_TYPE VALUES ('B','BAJA');
INSERT INTO HIST_ACTION_TYPE VALUES ('M','MODIFICACIÓN');
```

2.- Por cada tabla de la que se quiere conservar su información histórica se creará una tabla con los registros históricos, en adelante HIST. Por convenio del equipo la tabla HIST tendrá el nombre de la tabla original y el sufijo "*_HIST*" y estará formada por los siguientes campos:

- Los campos que forman la clave principal de la tabla original.
- Un campo timestamp, hace referencia al momento del cambio.
- Un campo que indique el tipo, hace referencia a ID_ACTION_TYPE.
- Los campos de la tabla principal de los que se quiera guardar información histórica.

4. ACCESO A LA APLICACIÓN

A la hora de diseñar el acceso a la aplicación se ha tenido en cuenta que ésta alberga datos de carácter personal, por lo que deberá cumplir la legislación actual LOPD (Ley Orgánica de Protección de Datos), para ello se han tomado las siguientes acciones:

1. **Proteger el acceso a la aplicación**.

Cómo método de protección al acceso a la aplicación se ha decidido la validación de usuario por contraseña.

El sistema deberá mantener en todo momento una relación en la que se determine qué usuario puede acceder a la aplicación, el nivel de acceso que tiene éste y la persona física asociada a dicho usuario.

2. **Protección de los datos**.

Como medida de protección de los datos se ha optado por guardar las contraseñas cifradas. El sistema de cifrado de contraseñas es configurable por el administrador de la aplicación por medio de la edición del fichero de configuración.

La protección del acceso físico a los datos, correrá a cargo del responsable de seguridad.

La protección del acceso a la base de datos estará protegido usando los medios de los que dispone el SGBD (Sistema Gestor de Bases de Datos) Oracle, es decir, mediante usuario y contraseña.

La protección al acceso lógico a la base de datos, correrá a cargo del administrador del sistema, el cual deberá implantar las siguientes soluciones:

- Acceso al SO (Sistema Operativo) mediante el uso de usuario y contraseña.
- Protección del fichero usando las herramientas que proporciona Windows 2000 como el EFS (Encrypted File System (Sistema de Ficheros Cifrados)) donde se usa la técnica de clave pública – clave privada.
- Otra herramienta que es útil dependiendo del Sistema de Ficheros del SO, será el uso de permisos NTFS, complementado con los permisos de carpeta compartida, en el caso de que se acceda a los ficheros a través de la red.

3. **Otros**

Según la LOPD, los datos que alberga la aplicación son de “Tipo Básico” por lo que no se ve necesario la implementación de sistemas de traza que permitan un seguimiento exhaustivo y control de lo que hace cada usuario en cada momento.

El fichero deberá darse de alta en la Agencia de Protección de Datos.

La creación del documento de seguridad, correrá a cargo del responsable del fichero.

Otras medidas contempladas en la LOPD, como puedan ser los sistemas de backup, almacenamiento de soportes, formularios de alta de usuarios, etc. correrá a cargo del administrador de la aplicación.

5. CONSIDERACIONES PARA EL DISEÑO

Del proceso de análisis se desprenden otras consideraciones que afectan al diseño del modelo.

5.1 Limitaciones

Se recomienda no exceder de 10 el número de campos por tabla, esta limitación viene dada por otros SGBD estudiados. Además, más de diez campos da idea de que la tabla debería normalizarse (dividirse) en varias tablas más pequeñas.

5.2 Nomenclatura

Respecto a la nomenclatura elegida se han seguido las siguientes reglas:

1. Los nombres asignados a las tablas y campos deberán ser lo más estándar posible por ello se ha elegido el inglés, por su carácter internacional y estándar actual en el mundo de la informática.
2. Los nombres deben ser claros y expresivos, aunque manteniendo una longitud máxima de 20 caracteres para facilitar la migración a otros SGBD.

   Se han dado casos en los que no ha sido posible seguir esta regla debido a la longitud que tomaba el nombre, por ello se ha omitido ciertas partes de los nombres de las tablas, quedándose con la parte más significativa de dichos nombres.
3. Ante la decisión de elegir nombres para las tablas, dependiendo del entorno de desarrollo, se pueden escribir en plural o en singular, eso sí, el criterio seleccionado será el que se deba seguir a lo largo de todo el desarrollo. En éste caso se ha decantado por los nombres en plural.
4. Las tablas que contengan históricos para diferenciarlas de las tablas “normales” se les añade la terminación *_HIST*, de éste modo serán identificadas a simple vista.
5. Las tablas creadas como relación de otras dos, se llamarán como el nombre de las dos tablas que relacionan concatenado por “_”.: *TABLA1_TABLA2*.
6. A la hora de escoger nombres para las claves principales, se usará el nombre de la tabla anexando al final *_PK*, de Primary_Key, de éste modo quedará perfectamente identificado a qué tabla hace referencia.
7. Para escoger nombres para las Claves Ajenas, se usará el nombre de las tablas a las que se hace alusión anexando al final *_FK*, de Foreign Key, de éste modo quedará perfectamente identificado a qué tablas hace referencia.
8. En aquellas tablas donde no haya ningún campo que pueda ser clave principal, se creará un campo “artificial”, cuyo nombre será el de la propia tabla precedido por ID_, con dicho campo luego se hará la Primary Key correspondiente.

6. Diseño del Modelo de Datos

6.1 Acceso a la Aplicación

Teniendo en cuenta toda la problemática anteriormente descrita se ha optado por implementar la seguridad a nivel de aplicación, y los datos estarán recogidos en las siguientes tablas:

INTRANET_USER: En ésta tabla se almacenan los distintos usuarios que pueden acceder a la aplicación, hay que tener en cuenta que también se almacenan las contraseñas que dichos usuarios usan, y que éstas se guardan cifradas. También se ha determinado añadir un campo a la tabla que nos indique el número de intentos de accesos erróneos que ha realizado el usuario, de éste modo cuando el usuario haya hecho tres fallidos se le bloquee la cuenta y tenga que contactar con el administrador, que será quien determine el problema.

HIST_ACTION_TYPE: En ésta tabla se registran los posibles movimientos que pueden tener los registros. Es una tabla útil a la hora de realizar los históricos, ya que será aquí donde se indique el significado de las acciones que se han realizado con los registros que se encuentran en las tablas de los históricos.

INTRANET_USER_HIST: Históricos de "INTRANET_USER". En ésta tabla se guardan los usuarios de la aplicación que han sido eliminados.

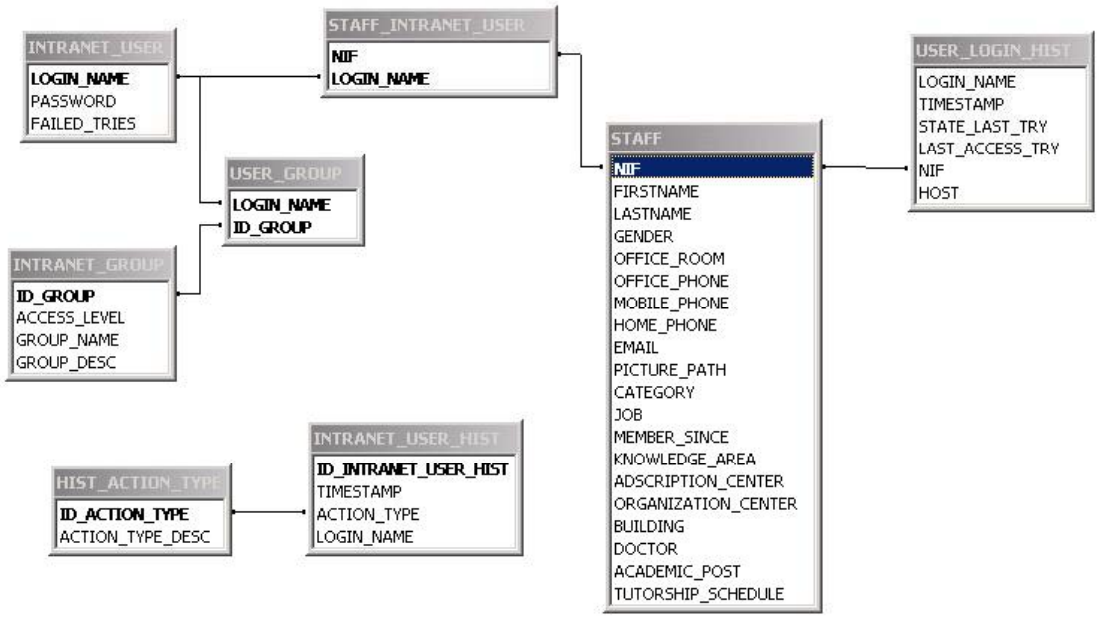
USER_LOGIN_HIST: En ésta tabla no se guarda ningún histórico de ninguna otra tabla, sino que se almacena información de los usuarios que usan la aplicación. Ésta tabla permite tener un control de los usuarios conectados, desde donde lo han hecho o cuanto tiempo hace desde que se conectara por última vez. Como histórico que es se ha seguido la normativa de añadir al final del nombre _HIST.

INTRANET_GROUP: Como se ha comentado en la descripción de la problemática, se precisan distintos niveles de acceso a la aplicación por ello se ha pensado en la creación de grupos de acceso, al igual que un sistema operativo (W2000, Linux, NT, Solaris, etc.). Por defecto la aplicación crea tres niveles de acceso: administrador, usuario avanzado y usuario, en caso de necesitar más variedad de niveles se deberán insertar en dicha tabla y darles en nivel de acceso correspondiente.

USER_GROUP: Al igual que en un sistema operativo un determinado usuario puede pertenecer a varios grupos, por ejemplo, el usuario *paco* puede pertenecer al grupo usuarios y luego por necesidades funcionales, se le podría insertar en el grupo usuarios avanzados. El sistema le dará privilegios conforme pertenezca a uno u otro grupo. Ésta tabla lo que hace es almacenar la relación existente entre usuarios y grupos a los que pertenecen.

USER_STAFF: En esta tabla se almacenan las relaciones entre usuario y personas físicas. Con ello se quiere tener un control de la relación entre usuario y persona física asociada a dicho usuario. Se podrían dar casos de que un persona física tuviera varios usuarios, esto es común en administradores o en personas que desempeñan varias funciones, de ésta manera dependiendo de la función que desarrolle dicha persona se validará en la aplicación con uno u otro usuario.

Gráficamente las tablas y sus relaciones quedarían del siguiente modo:



6.2 Personal

A la hora de diseñar la parte de personal, se ha tenido en cuenta los requisitos entregados, el modo de funcionamiento del departamento y la funcionalidad de la anterior aplicación que se detalla en el siguiente apartado.

Funcionalidades de la opción personal

La URL principal se sitúa a la izquierda, que cuando se pulsa con el ratón sobre ella se despliegan las siguientes posibilidades dependiendo del Rol o Perfil al que pertenezca el usuario que ha iniciado sesión:

Usuario:

- datos personales:

  Dentro de esta web aparecerán los datos del usuario que ha iniciado sesión en modo "editable" (exceptuando campos NO editables como el nivel de acceso o su identificador) y justo al pie, aparecerá un botón con el texto "modificar", para modificar los posibles cambios que se hayan hecho, y otro con el texto "salir" que nos llevaría a la pantalla anterior.

  Esta sección, por tanto, ofrece las funcionalidades de consulta y modificación de los datos personales simultáneamente.

- listados

  Dentro de ésta web aparecerán varios botones que realizan consultas estándar donde se le permita a cualquiera ver la información publica abreviada de los miembros del departamento (Número de teléfono, despacho, etc.)

  Posibles listados

  1. Ordenados por apellido:

Apellidos, nombre, tlf., departamento, despacho, asignatura

2. Ordenados por departamento:

Apellidos, nombre, tlf., departamento, despacho, asignatura

Usuario Avanzado, además de incluir las anteriores posibilidades ofrecerá también las siguientes:

- listados avanzados

Dentro de esta web aparecerán casillas donde se seleccionen los datos a listar, así como los campos por los cuales se quiere que se ordenen.

Administrador, además de incluir las anteriores posibilidades también ofrecerá las siguientes:

- gestión personal

Dentro de esta web aparecerá un listado (id, apellido1, apellido2, nombre..., esto puede variar) donde haciendo doble clic sobre un elemento de la lista se nos muestre otra web donde aparezcan los datos del ítem pulsado con las mismas características de la opción datos personales.

Además del anterior listado deberán aparecer otros tres botones:

- *alta* de personal que nos llevará a una web con un formulario que recoge los datos.

- *borrar* personal que eliminará el ítem seleccionado pidiendo confirmación.

- *cancelar* que nos lleva a la pantalla anterior.

- histórico

Dentro de ésta web aparecerá un listado con todos los registros que tenga el histórico, dicho listado podrá ser ordenado por los campos que muestre, además habrá dos botones uno para eliminar los registros seleccionados y otro de cancelar donde nos llevará a la pantalla anterior.

ESQUEMA GENERAL (desplegado para un usuario **administrador**)

Personal

Datos personales

Gestión personal

Listados

Listados avanzados

Históricos

ESQUEMA GENERAL (desplegado para un usuario **usuario avanzado**)

Personal

Datos personales

Listados

Listados avanzados

ESQUEMA GENERAL (desplegado para un usuario **usuario**)

Personal

Datos personales

Listados

Teniendo en cuenta toda la problemática y funcionalidad anteriormente descrita se han implementado las siguientes tablas:

STAFF: Ésta es la tabla principal de la aplicación ya que en ella se almacenan los datos del personal del departamento. Muchas otras tablas harán referencia al campo NIF de STAFF como Clave Extranjera (Foreign Key). Como curiosidad cabe destacar la longitud del NIF, que se ha escogido de 24, con el objetivo de hacerlo más internacional.

STAFF_INTRANET_USER: En esta tabla se relaciona los usuarios de la aplicación con el personal del departamento. Se puede dar el caso de que una misma persona física tenga varios usuarios en la aplicación, como también se puede dar el caso de que una persona física no tenga usuario de la aplicación. Se puede observar la tabla en el gráfico de "acceso a la aplicación".

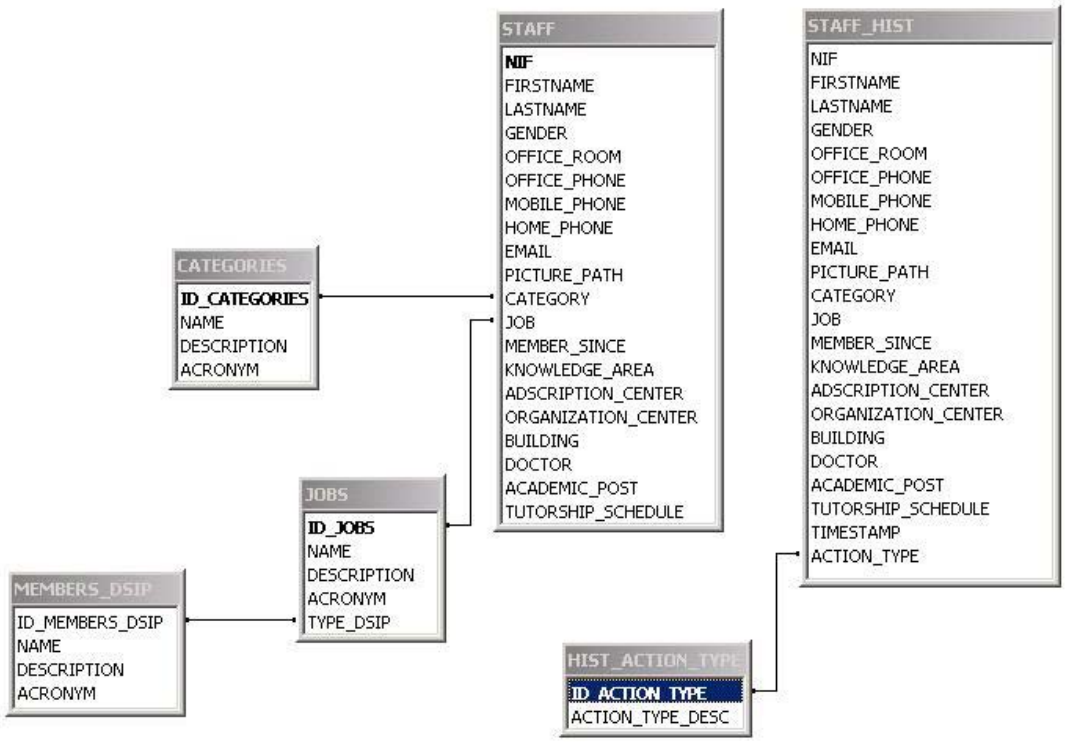
CATEGORIES: En ésta tabla se guarda las categorías a las que puede pertenecer el personal del departamento.

JOBS: En esta tabla se almacenan los distintos trabajos que desempeñan el personal del departamento.

MEMBERS_DSIP: En esta tabla se guarda los tipos de personal que hay en cada departamento, como puede ser DPI/PAS.

STAFF_HIST: Como su propia terminación (_HIST) indica, en ésta tabla se almacena un histórico de todo el personal que ha pertenecido al departamento, es decir, personal que ha sido eliminado por el administrador.

Gráficamente las tablas y sus relaciones quedarían del siguiente modo:


STAFF
NIF
FIRSTNAME
LASTNAME
GENDER
OFFICE_ROOM
OFFICE_PHONE
MOBILE_PHONE
HOME_PHONE
EMAIL
PICTURE_PATH
CATEGORY
JOB
MEMBER_SINCE
KNOWLEDGE_AREA
ADSCRIPTION_CENTER
ORGANIZATION_CENTER
BUILDING
DOCTOR
ACADEMIC_POST
TUTORSHIP_SCHEDULE
STAFF_HIST
NIF
FIRSTNAME
LASTNAME
GENDER
OFFICE_ROOM
OFFICE_PHONE
MOBILE_PHONE
HOME_PHONE
EMAIL
PICTURE_PATH
CATEGORY
JOB
MEMBER_SINCE
KNOWLEDGE_AREA
ADSCRIPTION_CENTER
ORGANIZATION_CENTER
BUILDING
DOCTOR
ACADEMIC_POST
TUTORSHIP_SCHEDULE
TIMESTAMP
ACTION_TYPE
CATEGORIES
ID_CATEGORIES
NAME
DESCRIPTION
ACRONYM
JOBS
ID_JOBS
NAME
DESCRIPTION
ACRONYM
TYPE_DSIP
MEMBERS_DSIP
ID_MEMBERS_DSIP
NAME
DESCRIPTION
ACRONYM
HIST_ACTION_TYPE
ID_ACTION_TYPE
ACTION_TYPE_DESC

7. CONSIDERACIONES PARA EL DESARROLLO

Ver documento “Optimización de bases de datos ORACLE”.

Ver documento “Acceso concurrente y Bases de Datos”.

8. IMPLANTACIÓN

8.1 Usuarios y Permisos en la Base de Datos

Como se ha comentado en la introducción del presente documento Oracle tiene herramientas y mecanismos para el control de acceso a los datos. Éstos mecanismos consisten en el acceso a datos previa identificación mediante usuario y contraseña.

A parte de los usuarios que vienen por defecto en el propio SGBD (SYSDBA y SYSOPER) se ha creado un usuario administrador (DBAUSER) de la base de datos creada (DSIP) para que genere la estructura de tablas necesarias para la aplicación y otro usuario operador (WEBUSER) que será el usuario que acceda vía web a la consulta, inserción, modificación y eliminación de registros.

El script que genera a los respectivos usuarios es el siguiente y lo deberá lanzar el usuario SYSDBA que es el único con permisos para crear usuarios por defecto. A continuación se detalla el script (en el parte IDENTIFIED BY no se ha puesto la contraseña por motivos de seguridad).

```
-- Usuario ADMINISTRADOR de la base de datos
-- User: DBAUSER          Password: intranetDSIP
CREATE USER DBAUSER
IDENTIFIED BY *****
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP;

GRANT DBA TO DBAUSER;

-- Usuario NORMAL (sólo acceso) a la base de datos
-- User: WEBUSER          Password: webuser
CREATE USER WEBUSER
IDENTIFIED BY *****
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP;

GRANT CONNECT, RESOURCE TO WEBUSER;
```

Cabe destacar el rol que se asigna a los usuarios:

- DBAUSER : Rol DBA, que le da todos los permisos para un administrador de base de datos.
- WEBUSER: Rol CONNECT, que le da todos los permisos necesarios para iniciar una sesión en Oracle.

Aunque Oracle permite la creación de roles que se adapten específicamente a la casuística de cada problema, se ha considerado que los roles de los que dispone Oracle son suficientes.

Para concretar aun más las restricciones de acceso sin perder funcionalidad se ha creado otro script que dará permisos específicos sobre cada una de las tablas.

Dicho script se detalla a continuación:

```
--LOGIN
GRANT SELECT ON HIST_ACTION_TYPE TO WEBUSER;
GRANT SELECT ON INTRANET_GROUP TO WEBUSER;
GRANT SELECT, INSERT, UPDATE, DELETE ON INTRANET_USER TO WEBUSER;
GRANT SELECT, INSERT, UPDATE ON INTRANET_USER_HIST TO WEBUSER;
GRANT SELECT, INSERT ON USER_GROUP TO WEBUSER;
GRANT SELECT, INSERT ON USER_LOGIN_HIST TO WEBUSER;

--PERSONAL
GRANT SELECT, INSERT, UPDATE, DELETE ON STAFF TO WEBUSER;
GRANT SELECT, INSERT ON STAFF_HIST TO WEBUSER;
GRANT SELECT, INSERT, UPDATE, DELETE ON CATEGORIES TO WEBUSER;
GRANT SELECT, INSERT, UPDATE, DELETE ON JOBS TO WEBUSER;
GRANT SELECT, INSERT, UPDATE, DELETE ON MEMBERS_DSIP TO WEBUSER;
GRANT SELECT, INSERT, UPDATE, DELETE ON STAFF_INTRANET_USER TO WEBUSER;
```

En donde:

SELECT permite la consulta de datos de la tabla.

INSERT permite la creación de registros en la tabla correspondiente.

UPDATE da la posibilidad de actualizaciones o modificaciones sobre registros.

DELETE da los permisos de eliminación de registros dentro de la tabla seleccionada.

8.2 Uso de Sinónimos

El procedimiento lógico en la creación de la base de datos es el siguiente:

1. El DBA (Data Base Administrator) que viene por defecto (SYSDBA) crea usuarios con distintos roles dentro de la BBDD.
2. Uno de los usuarios creado debe tener permisos de Administrador para crear, modificar y eliminar tablas, en nuestro caso DBAUSER.
3. Otro de los usuarios creado debe tener permisos de Usuario normal, que consulta, inserta, modifica y elimina registros de las tablas, en nuestro caso WEBUSER.

El problema viene cuando el usuario DBAUSER, crea la estructura de tablas, y luego el usuario WEBUSER, intenta acceder a tablas que no ha creado él, con lo que se encuentra con un mensaje parecido a este "TABLA O VISTA DESCONOCIDA".

Una solución sería acceder a la tabla anteponiendo su creador del siguiente modo: “SELECT * FROM DBAUSER.TABLA”, pero esto añadiría la restricción de que todos los usuarios deban saber que la tabla pertenece a un usuario en concreto. Hay una solución más elegante: el uso de Sinónimos.

Los sinónimos son una peculiaridad de Oracle (y de otros SGBD´s) que permite establecer un segundo nombre (en este caso público, para todos lo demás usuarios) de las tablas.

A continuación de detalla el script que genera los sinónimos necesarios en la base de datos para que cualquier usuario acceda a las tablas sin problemas.

```
CREATE PUBLIC SYNONYM CATEGORIES FOR DBAUSER.CATEGORIES;
CREATE PUBLIC SYNONYM HIST_ACTION_TYPE FOR DBAUSER.HIST_ACTION_TYPE;
CREATE PUBLIC SYNONYM INTRANET_GROUP FOR DBAUSER.INTRANET_GROUP;
CREATE PUBLIC SYNONYM INTRANET_USER FOR DBAUSER.INTRANET_USER;
CREATE PUBLIC SYNONYM INTRANET_USER_HIST FOR DBAUSER.INTRANET_USER_HIST;
CREATE PUBLIC SYNONYM JOBS FOR DBAUSER.JOBS;
CREATE PUBLIC SYNONYM MEMBERS_DSIP FOR DBAUSER.MEMBERS_DSIP;
CREATE PUBLIC SYNONYM STAFF FOR DBAUSER.STAFF;
CREATE PUBLIC SYNONYM STAFF_HIST FOR DBAUSER.STAFF_HIST;
CREATE PUBLIC SYNONYM STAFF_INTRANET_USER FOR DBAUSER.STAFF_INTRANET_USER;
CREATE PUBLIC SYNONYM USER_GROUP FOR DBAUSER.USER_GROUP;
CREATE PUBLIC SYNONYM USER_LOGIN_HIST FOR DBAUSER.USER_LOGIN_HIST;
```

Bibliografía

Las webs oficiales de ORACLE:
http://www.oracle.com/
http://otn.oracle.com/

Curso de iniciación de ORACLE:
http://www.lawebdejm.com/prog/oracle

Tutoriales de SQL y SGBD:
http://www.tutoriales.com
http://www.programacion.com/direcciones/bases-de-datos/

Reglamento de la Ley Orgánica de Protección de Datos (LOPD):
http://www.valorweb.com/descargas/lopd.pdf

Apéndice A

En estos scripts esta recogida la creación del modelo de datos.

Script SQL (Acceso a la Aplicación)

```
/*************************************************************************************************/
/*      Tabla INTRANET_USER: Usuarios que tienen acceso a la aplicación                  */

CREATE TABLE INTRANET_USER (
 LOGIN_NAME              VARCHAR2(16)     NOT NULL,
 PASSWORD     VARCHAR2(16)     NOT NULL,
 FAILED_TRIES            NUMBER                   DEFAULT 0 CHECK (FAILED_TRIES<4)
);

/* Clave primaria de INTRANET_USER                                        */
ALTER TABLE INTRANET_USER ADD (
      CONSTRAINT INTRANET_USER_PK
            PRIMARY KEY      (LOGIN_NAME)
);

/*      Comentarios de la tabla INTRANET_USER                                                    */
COMMENT ON TABLE  INTRANET_USER     IS 'Tabla que almacena los usuarios que utilizarán la aplicación';
COMMENT ON COLUMN INTRANET_USER.LOGIN_NAME IS 'PK, Identificador único del usuario de la INTRANET';
COMMENT ON COLUMN INTRANET_USER.PASSWORD IS 'Password encriptada y personal de cada usuario';
COMMENT ON COLUMN INTRANET_USER.FAILED_TRIES IS 'Número de intentos fallidos de accesos (Se resetea a 0 cuando el acceso ha sido exitoso)';


/*************************************************************************************************/
/*      Tabla HIST_ACTION_TYPE: Posibles acciones que se pueden hacer con los registros      */

CREATE TABLE HIST_ACTION_TYPE (
 ID_ACTION_TYPE       VARCHAR(1)       UNIQUE NOT NULL,
 ACTION_TYPE_DESC     VARCHAR2(15)     NOT NULL
);

/*      Comentarios de la tabla HIST_ACTION_TYPE                                                */
COMMENT ON TABLE HIST_ACTION_TYPE IS 'Tabla que almacena las posibles acciones que se han efectuado con los registros y han almacenado en el histórico';
COMMENT ON COLUMN HIST_ACTION_TYPE.ID_ACTION_TYPE IS 'A, Alta. B, Baja. M, Modificacion';


/*************************************************************************************************/
/*      Tabla INTRANET_USER_HIST: Histórico de usuarios de la aplicación                 */

CREATE TABLE INTRANET_USER_HIST (
 ID_INTRANET_USER_HIST          NUMBER           NOT NULL,
 TIMESTAMP                DATE             DEFAULT SYSDATE,
 ACTION_TYPE                    VARCHAR(1)       NOT NULL,
 LOGIN_NAME               VARCHAR2(16)     NOT NULL
);

/* Clave primaria de INTRANET_USER_HIST                                       */
ALTER TABLE INTRANET_USER_HIST ADD (
      CONSTRAINT INTRANET_USER_HIST_PK
            PRIMARY KEY      (ID_INTRANET_USER_HIST)
```

```
);

/*      Comentarios de la tabla INTRANET_USER_HIST                                      */
COMMENT ON TABLE INTRANET_USER_HIST IS 'Histórico de las acciones realizadas sobre INTRANET_USER';
COMMENT ON COLUMN INTRANET_USER_HIST.TIMESTAMP IS 'Instante en el que se produce la ACTION_TYPE';
COMMENT ON COLUMN INTRANET_USER_HIST.ACTION_TYPE IS 'Acción que se produce con el usuario: A, Alta. B, Baja. M, Modificacion';
COMMENT ON COLUMN INTRANET_USER_HIST.LOGIN_NAME IS 'Usuario que ha sido alterado';


/*************************************************************************************************/
/*      Tabla USER_LOGIN_HIST: Histórico de los usuarios validados en la aplicación     */

CREATE TABLE USER_LOGIN_HIST (
 LOGIN_NAME                 VARCHAR2(16) NOT NULL,
 TIMESTAMP          DATE            DEFAULT SYSDATE,
 STATE_LAST_TRY     VARCHAR(1)      DEFAULT '0',
 LAST_ACCESS_TRY    DATE            DEFAULT SYSDATE,
 NIF                VARCHAR2(24) NOT NULL,
 HOST                       VARCHAR(15)     DEFAULT NULL
);

/* Clave primaria de USER_LOGIN_HIST                                    */
ALTER TABLE USER_LOGIN_HIST ADD (
        CONSTRAINT USER_LOGIN_HIST_PK
                PRIMARY KEY     (LOGIN_NAME,TIMESTAMP)
);

/*      Comentarios de la tabla USER_LOGIN_HIST                                         */
COMMENT ON TABLE  USER_LOGIN_HIST IS 'Almacena lo usuarios que se han ido logando en la aplicación';
COMMENT ON COLUMN USER_LOGIN_HIST.LOGIN_NAME IS 'PK, Usuario relacionado con DSIP.USER';
COMMENT ON COLUMN USER_LOGIN_HIST.TIMESTAMP IS 'PK, Instante de tiempo en el que se registra';
COMMENT ON COLUMN USER_LOGIN_HIST.STATE_LAST_TRY IS 'Estado del último intento de acceder a la INTRANET (0 - FALLO, 1-EXITO)';
COMMENT ON COLUMN USER_LOGIN_HIST.LAST_ACCESS_TRY IS 'Fecha del último intento de acceder o acceso a la INTRANET';
COMMENT ON COLUMN USER_LOGIN_HIST.NIF IS 'IDENTIFICADOR NACIONAL INTERNACIONALIZADO';
COMMENT ON COLUMN USER_LOGIN_HIST.HOST IS 'IP de la máquina donde se realizó el último intento';


/*************************************************************************************************/
/*      Tabla INTRANET_GROUP: Grupos de nivel de acceso a la aplicación         */

CREATE TABLE INTRANET_GROUP (
 ID_GROUP       NUMBER NOT NULL,
 ACCESS_LEVEL           NUMBER NOT NULL CHECK (ACCESS_LEVEL>=0),
 GROUP_NAME             VARCHAR2(20)       NOT NULL,
 GROUP_DESC VARCHAR2(256)
);

/* Clave primaria de INTRANET_GROUP                                     */
ALTER TABLE INTRANET_GROUP ADD (
        CONSTRAINT INTRANET_GROUP_PK
                PRIMARY KEY     (ID_GROUP)
);

/*      Comentarios de la tabla INTRANET_GROUP                                          */
```

```
COMMENT ON TABLE  INTRANET_GROUP IS 'Tabla que almacena los grupos de nivel al que pueden pertenecer los usuarios';
COMMENT ON COLUMN INTRANET_GROUP.ID_GROUP IS 'PK, Identificador del grupo';
COMMENT ON COLUMN INTRANET_GROUP.ACCESS_LEVEL IS 'Nivel que define el acceso de este grupo. Varios grupos pueden tener los mismo niveles';
COMMENT ON COLUMN INTRANET_GROUP.GROUP_NAME IS 'Nombre del grupo';


/*************************************************************************************************/
/*      Tabla USER_GROUP: Relaciona usuarios de la aplicación con el grupo al que pertenecen */
CREATE TABLE USER_GROUP (
 LOGIN_NAME             VARCHAR2(16)     NOT NULL,
 ID_GROUP     NUMBER             NOT NULL
);

/* Clave primaria de USER_GROUP                                          */
ALTER TABLE USER_GROUP ADD (
        CONSTRAINT USER_GROUP_PK
                PRIMARY KEY      (LOGIN_NAME,ID_GROUP)
);

/*      Comentarios de la tabla INTRANET_GROUP                                  */
COMMENT ON TABLE USER_GROUP IS 'Define los grupos (nivel de acceso) a los que pertenece cada usuario de la aplicación';
COMMENT ON COLUMN USER_GROUP.LOGIN_NAME IS 'Usuario de la aplicación';
COMMENT ON COLUMN USER_GROUP.ID_GROUP IS 'Grupo o nivel de acceso que tiene en la aplicación';
```

Script SQL (Personal)

```
/*************************************************************************************************/
/*      Tabla STAFF: Personal gestionado por la intranet                */

CREATE TABLE STAFF (
        NIF                 VARCHAR2(24)     NOT NULL,
        FIRSTNAME           VARCHAR2(50)     NOT NULL,
        LASTNAME            VARCHAR2(50)     NOT NULL,
        GENDER              VARCHAR(1)       NOT NULL CHECK (GENDER='H' OR GENDER='M'),
        OFFICE_ROOM         VARCHAR2(10),
        OFFICE_PHONE        VARCHAR2(12),
        MOBILE_PHONE        VARCHAR2(12),
        HOME_PHONE          VARCHAR2(12),
        EMAIL               VARCHAR2(50),
        PICTURE_PATH        VARCHAR2(256),
        CATEGORY            NUMBER,
        JOB                 NUMBER,
        MEMBER_SINCE        DATE,
        KNOWLEDGE_AREA          VARCHAR2(50),
        ADSCRIPTION_CENTER      VARCHAR2(20),
        ORGANIZATION_CENTER     VARCHAR2(20),
        BUILDING            VARCHAR2(10),
        DOCTOR              VARCHAR(1),
        ACADEMIC_POST       VARCHAR2(20),
        TUTORSHIP_SCHEDULE      VARCHAR2(40)
);

/* Clave primaria de STAFF                                       */
ALTER TABLE STAFF ADD (
```

```
        CONSTRAINT STAFF_PK
                PRIMARY KEY (NIF)
    );

    /* Insertamos los comentarios en la tabla STAFF                                    */
    COMMENT ON TABLE STAFF IS 'Tabla que almacena el personal del departamento';
    COMMENT ON COLUMN STAFF.NIF IS 'NIF de la persona, además es la clave de la tabla';
    COMMENT ON COLUMN STAFF.GENDER IS 'H->Hombre o M->Mujer';
    COMMENT ON COLUMN STAFF.OFFICE_ROOM IS 'Despacho que ocupa la persona';
    COMMENT ON COLUMN STAFF.PICTURE_PATH IS 'Ruta que alberga la foto de la persona';
    COMMENT ON COLUMN STAFF.CATEGORY IS 'Código de la categoría de trabajo que tiene la persona';
    COMMENT ON COLUMN STAFF.JOB IS 'Código del trabajo que ejerce la persona';
    COMMENT ON COLUMN STAFF.MEMBER_SINCE IS 'Fecha de nombramiento o contratación en su categoría';
    COMMENT ON COLUMN STAFF.KNOWLEDGE_AREA IS 'Área de conocimiento, sólo profesores, LSI o CCIA';
    COMMENT ON COLUMN STAFF.ADSCRIPTION_CENTER IS 'Centro de adscripción, oficial';
    COMMENT ON COLUMN STAFF.ORGANIZATION_CENTER IS 'Centro de organización, actualmente dos posibilidades:
FI -núcleo central o FCM';
    COMMENT ON COLUMN STAFF.BUILDING IS 'Edificio donde está situado el despacho (office_room)';
    COMMENT ON COLUMN STAFF.DOCTOR IS 'Valor booleano que indica si la persona es doctor';
    COMMENT ON COLUMN STAFF.ACADEMIC_POST IS 'Cargo académico que obstenta la persona';
    COMMENT ON COLUMN STAFF.TUTORSHIP_SCHEDULE IS 'Horario de tutorías';


    /*************************************************************************************************/
    /*      Tabla STAFF_INTRANET_USER: Relaciona personal del departamento con los usuarios      */

    CREATE TABLE STAFF_INTRANET_USER (
        NIF             VARCHAR2(24)    NOT NULL,
        LOGIN_NAME      VARCHAR2(16)    NOT NULL
    );

    /* Clave primaria de STAFF_INTRANET_USER                                   */
    ALTER TABLE STAFF_INTRANET_USER ADD (
        CONSTRAINT STAFF_INTRANET_USER_PK
                PRIMARY KEY     (NIF,LOGIN_NAME)
    );

    /* Insertamos los comentarios en la tabla STAFF_INTRANET_USER                                    */
    COMMENT ON TABLE STAFF_INTRANET_USER IS 'Tabla que relaciona personal del departamento con los usuarios de la
aplicación';
    COMMENT ON COLUMN STAFF_INTRANET_USER.NIF IS 'NIF de la persona del departamento';
    COMMENT ON COLUMN STAFF_INTRANET_USER.LOGIN_NAME IS 'Login o nombre de usuario asociado a la persona
puede haber más de uno';


    /*************************************************************************************************/
    /*      Tabla CATEGORIES: categorías a las que puede pertenecer el personal dsip        */

    CREATE TABLE CATEGORIES (
        ID_CATEGORIES   NUMBER NOT NULL,
        NAME            VARCHAR2(50),
        DESCRIPTION     VARCHAR2(100),
        ACRONYM                 VARCHAR2(10)
    );

    /* Clave primaria de CATEGORIES                                    */
    ALTER TABLE CATEGORIES ADD (
        CONSTRAINT CATEGORIES_PK
```

```
        PRIMARY KEY (ID_CATEGORIES)
);

/* Insertamos los comentarios en la tabla CATEGORIES                                    */
COMMENT ON TABLE CATEGORIES IS 'Tabla que almacena las categorías a las que puede pertenecer el personal del departamento';
COMMENT ON COLUMN CATEGORIES.NAME IS 'Nombre de la categoría';
COMMENT ON COLUMN CATEGORIES.ACRONYM IS 'Iniciales que definen a la categoría';


/*************************************************************************************************/
/*      Tabla JOBS: Trabajos que desempeñan el personal de DSIP                 */

CREATE TABLE JOBS (
        ID_JOBS         NUMBER NOT NULL,
        NAME            VARCHAR2(50),
        DESCRIPTION     VARCHAR2(100),
        ACRONYM                 VARCHAR2(10),
        TYPE_DSIP       NUMBER
);

/* Clave primaria de JOBS                                       */
ALTER TABLE JOBS ADD (
        CONSTRAINT JOBS_PK
                PRIMARY KEY (ID_JOBS)
);

/* Insertamos los comentarios en la tabla JOBS                                  */
COMMENT ON TABLE JOBS IS 'Tabla que almacena las categorías a las que puede pertenecer el personal del departamento';
COMMENT ON COLUMN JOBS.NAME IS 'Nombre del trabajo';
COMMENT ON COLUMN JOBS.ACRONYM IS 'Iniciales que definen al trabajo';
COMMENT ON COLUMN JOBS.TYPE_DSIP IS 'Código que identifica a qué tipo del departamento pertenece el trabajo (DPI/PAS)';


/*************************************************************************************************/
/*      Tabla MEMBERS_DSIP: Tipos de personal de hay en el departamento (DSIP)          */

CREATE TABLE MEMBERS_DSIP (
        ID_MEMBERS_DSIP         NUMBER NOT NULL,
        NAME            VARCHAR2 (20),
        DESCRIPTION     VARCHAR2 (100),
        ACRONYM                 VARCHAR2 (10)
);

/* Clave primaria de MEMBERS_DSIP                                                       */
ALTER TABLE MEMBERS_DSIP ADD (
        CONSTRAINT MEMBERS_DSIP_PK
                PRIMARY KEY (ID_MEMBERS_DSIP)
);

/* Insertamos los comentarios en la tabla MEMBERS_DSIP                                  */
COMMENT ON TABLE MEMBERS_DSIP IS 'Tabla que almacena los tipos de personal que podemos encontrarnos en el departamento (DPI/PAS)';
COMMENT ON COLUMN MEMBERS_DSIP.ID_MEMBERS_DSIP IS 'Clave principal de la tabla';
COMMENT ON COLUMN MEMBERS_DSIP.NAME IS 'Nombre del tipo de personal del departamento';
COMMENT ON COLUMN MEMBERS_DSIP.ACRONYM IS 'Iniciales que definen al tipo de personal del departamento';
```

```
/*************************************************************************************************/
/*      Tabla STAFF_HIST: Histórico del personal gestionado por la intranet              */

CREATE TABLE STAFF_HIST (
        NIF                 VARCHAR2(24)    NOT NULL,
        FIRSTNAME           VARCHAR2(50)    NOT NULL,
        LASTNAME            VARCHAR2(50)    NOT NULL,
        GENDER              VARCHAR(1)      NOT NULL CHECK (GENDER='H' OR GENDER='M'),
        OFFICE_ROOM         VARCHAR2(10),
        OFFICE_PHONE        VARCHAR2(12),
        MOBILE_PHONE        VARCHAR2(12),
        HOME_PHONE          VARCHAR2(12),
        EMAIL               VARCHAR2(50),
        PICTURE_PATH        VARCHAR2(256),
        CATEGORY            VARCHAR2(50),
        JOB                 vARCHAR2(50),
        MEMBER_SINCE        DATE,
        KNOWLEDGE_AREA          VARCHAR2(50),
        ADSCRIPTION_CENTER      VARCHAR2(20),
        ORGANIZATION_CENTER     VARCHAR2(20),
        BUILDING            VARCHAR2(10),
        DOCTOR              VARCHAR(1),
        ACADEMIC_POST       VARCHAR2(20),
        TUTORSHIP_SCHEDULE      VARCHAR2(40),
        TIMESTAMP           DATE                DEFAULT SYSDATE,
        ACTION_TYPE         VARCHAR(1)      NOT NULL
);

/* Clave primaria de STAFF_HIST                                         */
ALTER TABLE STAFF_HIST ADD (
        CONSTRAINT STAFF_HIST_PK
                PRIMARY KEY (NIF, TIMESTAMP)
);

/* Insertamos los comentarios en la tabla STAFF_HIST                                    */
COMMENT ON TABLE STAFF_HIST IS 'Tabla que almacena el personal del departamento';
COMMENT ON COLUMN STAFF_HIST.NIF IS 'NIF de la persona, además es la clave de la tabla';
COMMENT ON COLUMN STAFF_HIST.GENDER IS 'H->Hombre o M->Mujer';
COMMENT ON COLUMN STAFF_HIST.OFFICE_ROOM IS 'Despacho que ocupa la persona';
COMMENT ON COLUMN STAFF_HIST.PICTURE_PATH IS 'Ruta que alberga la foto de la persona';
COMMENT ON COLUMN STAFF_HIST.CATEGORY IS 'Nombre de la categoría de trabajo que tiene la persona';
COMMENT ON COLUMN STAFF_HIST.JOB IS 'Nombre del trabajo que ejerce la persona';
COMMENT ON COLUMN STAFF_HIST.MEMBER_SINCE IS 'Fecha de nombramiento o contratación en su categoría';
COMMENT ON COLUMN STAFF_HIST.KNOWLEDGE_AREA IS 'Área de conocimiento, sólo profesores, LSI o CCIA';
COMMENT ON COLUMN STAFF_HIST.ADSCRIPTION_CENTER IS 'Centro de adscripción, oficial';
COMMENT ON COLUMN STAFF_HIST.ORGANIZATION_CENTER IS 'Centro de organización, actualmente dos posibilidades: FI -núcleo central o FCM';
COMMENT ON COLUMN STAFF_HIST.BUILDING IS 'Edificio donde está situado el despacho (office_room)';
COMMENT ON COLUMN STAFF_HIST.DOCTOR IS 'Valor booleano que indica si la persona es doctor';
COMMENT ON COLUMN STAFF_HIST.ACADEMIC_POST IS 'Cargo académico que obstenta la persona';
COMMENT ON COLUMN STAFF_HIST.TUTORSHIP_SCHEDULE IS 'Horario de tutorías';
COMMENT ON COLUMN STAFF_HIST.TIMESTAMP IS 'Instante en el que se produce la alteración de la persona';
COMMENT ON COLUMN STAFF_HIST.ACTION_TYPE IS 'Acción que se produce con el usuario: A, Alta. B, Baja. M, Modificacion';
```

Apéndice B

A continuación se detalla el script que genera la integridad referencial:

```
/*==============================================================================================*/
/*                                                                                              */
/*      Exigimos que en las tablas de históricos las acciones que se hagan estén en la tabla    */
/*      ACTION_TYPE_HIST                                                                        */
/*                                                                                              */
/*==============================================================================================*/
ALTER TABLE INTRANET_USER_HIST ADD (
        CONSTRAINT USER_HIST_ACTION_TYPE_HIST_FK
                FOREIGN KEY (ACTION_TYPE) REFERENCES HIST_ACTION_TYPE (ID_ACTION_TYPE)
);

ALTER TABLE STAFF_HIST ADD (
        CONSTRAINT STAFF_HIST_ACTION_TYPE_HIST_FK
                FOREIGN KEY (ACTION_TYPE) REFERENCES HIST_ACTION_TYPE (ID_ACTION_TYPE)
);


/*==============================================================================================*/
/*                                                                                              */
/*      Exigimos que en los campos category y job de la tabla STAFF tengan su correspondencia   */
/*      en las tablas CATEGORIES y JOBS                                                         */
/*                                                                                              */
/*==============================================================================================*/
ALTER TABLE STAFF ADD (
        CONSTRAINT STAFF_CATEGORIES_FK
                FOREIGN KEY (CATEGORY) REFERENCES CATEGORIES (ID_CATEGORIES)
);

ALTER TABLE STAFF ADD (
        CONSTRAINT STAFF_JOBS_FK
                FOREIGN KEY (JOB) REFERENCES JOBS (ID_JOBS)
);


/*==============================================================================================*/
/*                                                                                              */
/*      Exigimos que en el campo type_dsip de la tabla JOBS tenga su correspondencia con la     */
/*      tabla MEMBERS_DSIP                                                                      */
/*                                                                                              */
/*==============================================================================================*/
ALTER TABLE JOBS ADD (
        CONSTRAINT JOBS_MEMBERS_DSIP_FK
                FOREIGN KEY (TYPE_DSIP) REFERENCES MEMBERS_DSIP (ID_MEMBERS_DSIP)
);


/*==============================================================================================*/
/*                                                                                              */
/*      Exigimos que la tabla USER_GROUP que relaciona usuarios de la aplicación con los        */
/*      grupos de accesos tengan en sus campos datos validados en las tablas INTRANET_USER      */
/*      e INTRANET_GROUP, teniendo en cuenta que un mismo usuario puede pertenecer a varios     */
/*      grupos                                                                                  */
```

```
/*                                                                                                    */
/******************************************************************************************************/
ALTER TABLE USER_GROUP ADD (
        CONSTRAINT USER_GROUP_INTRANET_USER_FK
                FOREIGN KEY (LOGIN_NAME) REFERENCES INTRANET_USER (LOGIN_NAME)
);

ALTER TABLE  USER_GROUP ADD (
        CONSTRAINT USER_GROUP_INTRANET_GROUP_FK
                FOREIGN KEY (ID_GROUP) REFERENCES INTRANET_GROUP (ID_GROUP)
);


/******************************************************************************************************/
/*                                                                                                    */
/*      Exigimos que la tabla USER_STAFF que relaciona usuarios de la aplicación con las      */
/*      personas del departamento tengan en sus campos datos validados en las tablas STAFF    */
/*      e INTRANET_USER, teniendo en cuenta que una misma persona puede tener varios usuarios  */
/*                                                                                                    */
/******************************************************************************************************/
ALTER TABLE STAFF_INTRANET_USER ADD (
        CONSTRAINT STAFF_INTRANET_USER_USER_FK
                FOREIGN KEY (LOGIN_NAME) REFERENCES INTRANET_USER (LOGIN_NAME)
);

ALTER TABLE STAFF_INTRANET_USER ADD (
        CONSTRAINT STAFF_INTRANET_STAFF_FK
                FOREIGN KEY (NIF) REFERENCES STAFF (NIF)
);
```

Apéndice C

En estos scripts esta recogida la inserción de los datos por defecto.

Para el correcto funcionamiento de la aplicación son necesarios algunos datos que deberán cargarse previamente en la base de datos, antes de ser usada.

Estos datos principalmente hacen referencia a dos aspectos:

1. A las acciones permitidas, es decir, altas, bajas y modificaciones.
2. Los tipos de nivel de acceso que hay en la aplicación, es decir, Administrador, Usuario Avanzado y Usuario

A continuación se muestra el script SQL que carga dichos datos.

```
/*********************************************************************************************/
/*                                                                                           */
/*      Datos iniciales de la aplicación                                                     */
/*                                                                                           */
/*      Se recomienda NO alterar dichos valores                                              */
/*                                                                                           */
/*********************************************************************************************/

/* Valores para guardar los históricos                                                       */
INSERT INTO HIST_ACTION_TYPE VALUES ('A','ALTA');
INSERT INTO HIST_ACTION_TYPE VALUES ('B','BAJA');
INSERT INTO HIST_ACTION_TYPE VALUES ('M','MODIFICACIÓN');

/* Valores de los roles o grupos que están definidas en la aplicación                        */
INSERT INTO INTRANET_GROUP VALUES (1,0,'Administradores','Grupo Administradores, gestiona TODAS las posibilidades de la aplicación');
INSERT INTO INTRANET_GROUP VALUES (2,1,'Usuarios Avanzados','Grupo Usuarios Avanzados, tiene ciertos permisos en algunas partes de la aplicación');
INSERT INTO INTRANET_GROUP VALUES (3,2,'Usuarios','Grupo Usuarios, tiene permisos de consulta y otros accesos simples');
```

Enterprise Java Beans

Intranet SIP

25 de Junio de 2004, v1.0

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

1. PRÓLOGO

El propósito de este documento es el de resumir las características de la tecnología Enterprise Java Beans de Sun Microsystems y como se ha aplicado a la Intranet de Departamento de Sistemas Informáticos y Programación.

2. INTRODUCCIÓN

La tecnología EJB se engloba dentro del conjunto J2EE (Java 2 Enterprise Edition) de tecnologías empresariales basadas en Java

Esta compuesta por una serie de componentes de la capa de Negocio que aportan una plataforma de conectividad estándar con la capa de Datos en una aplicación web de arquitectura multicapa. Asimismo garantizan la escalabilidad y extensibilidad de la aplicación y la seguridad de los datos.

Sun proporciona esta definición de EJBs: "Los Enterprise Java Beans se definen como componentes distribuidos orientados a transacciones que permiten construir aplicaciones empresariales".

3. TIPOS DE EJBs

Los EJBs se dividen en Beans de Sesión y Beans de Entidad.

3.1 Beans de Sesión

Los Beans de Sesión se crean específicamente para un cliente, que guardará una referencia para acceder a él. No permiten acceso concurrente y su ciclo de vida lo controla el cliente. Se dividen a su vez en Beans de Sesión Con Estado (Stateful) o Sin Estado (Stateless).

3.1.1 EJBs de Sesión con Estado

Formados por un conjunto de métodos de negocio para interactuar con la capa de datos de la aplicación y una serie de atributos que permiten mantener información del Estado, es decir, una serie de variables que guardan información sobre las ultimas interacciones del cliente con la capa de datos.

Un hecho derivado de éste es que si el servidor necesita la memoria que tienen ocupada los Beans con Estado, entonces esa información deberá ser guardada en disco de modo que el estado de la sesión pueda ser recuperado en futuras llamadas. Es la llamada Persistencia de los EJBs. Normalmente el mecanismo utilizado es la Serialización de Java aunque pueden utilizarse otros mecanismos.

3.1.2 EJBs de Sesión sin Estado

Solo están formados por una serie de métodos de negocio que interactúan con la capa de datos. No mantienen información entre múltiples llamadas.

Si el período máximo de inactividad marcado por el servidor se excede, entonces la instancia del Bean de Sesión sin Estado puede ser enviada a un Pool de Beans inactivos o eliminada de memoria. Si el cliente realiza otra llamada, el servidor puede reactivar la instancia original desde el Pool o crear otra instancia.

Parece lógico pensar que dado que no se guarda información alguna entre llamadas, los EJB sin Estado no deberían tener Atributos de Clase, sino únicamente Métodos que puedan ser invocados por el cliente que ha solicitado el Bean.

Como se puede ver por esta definición, un EJB de Sesión sin estado no es mas que una interfaz común de acceso a datos. Se pueden crear EJBs sin Estado que representen Interacciones con una única tabla de la Base de Datos o con un conjunto cerrado e interrelacionado de tablas, con lo que se consigue un nivel de abstracción mayor sobre la capa de datos. Con esto se puede conseguir una mayor organización entre los programadores y los diseñadores del modelo, se establecen una serie de pasarelas o interfaces fijas de obtención/modificación de datos a través de los EJBs que facilitan el trabajo de ambas partes.

Como veremos, se corresponden en gran medida con el patrón de diseño Data Access Object (Objeto de Acceso a Datos), y es de esta manera como va a ser utilizado en el proyecto de la Intranet del DSIP:

Los EJBs de Sesión sin Estado serán usados a modo de DAOs. Se creará un Pool de DAOs (EJBs) al arrancar la aplicación, y estos serán compartidos por todos los clientes.

Cuando un cliente necesite acceder a la capa de datos ya sea para una consulta o modificación, deberá solicitar un EJB del Pool. Si hay EJBs libres se le asignará uno durante el tiempo de la interacción. Cuando se termine la operación con la capa de Datos, el EJB es devuelto al Pool para su posterior uso por otros clientes. Dado que sólo contienen métodos de negocio, no mantienen datos entre diferentes llamadas de clientes y son por tanto seguros en este aspecto.

A simple vista se puede ver la similitud con los Pools de conexiones a base de datos, pero en realidad los EJBs son una capa por encima de las conexiones. Con este tipo de EJBs se consiguen dos nuevos beneficios claros, aparte de los beneficios de organización y abstracción que se han citado anteriormente:

- *Eliminación de cuellos de botella* en el acceso a los DAO. Normalmente en las antiguas aplicaciones empresariales se tenía una sola instancia de una clase que permitía el acceso a los datos. Las peticiones de los clientes se atendían en orden y por tanto se creaban cuellos de botella. Gracias a que se puede crear un Pool de EJBs se elimina este riesgo.
- *Mayor rapidez*. Los Objetos EJB ya están creados desde el inicio de la aplicación, por lo que sólo se necesita obtener una referencia a ellos.

3.2 Beans de Entidad

Los Beans de Entidad representan una vista de Orientada a Objetos de los datos almacenados en una base de datos relacional, o cualquier otro método de almacenamiento necesariamente orientado a Objetos.

Son Persistentes y Transaccionales, es decir:

- Un Bean de Entidad representa un objeto persistente por naturaleza, como por ejemplo una registro en una tabla de una base de datos.
- Son transaccionales, por lo que cualquier operación sobre los datos forma parte necesariamente de una transacción, con lo que se salvaguarda la integridad de los datos.

La persistencia de un Bean puede ser controlada por el Contenedor (Máquina Virtual de Java sobre la que se ejecuta el servidor) o por sí mismo (mediante métodos implementados a ese efecto). En el primer caso se dice que existe CMP (Container Managed Persistence) y en el segundo se dice que hay BMP (Bean Managed Persistence).

Además son distribuidos y por tanto podrían ser compartidos por varios servidores de aplicaciones que formasen un Cluster (como se usan normalmente para aportar alta disponibilidad). Un cliente podría conectarse sucesivas veces contra diferentes servidores de aplicaciones en Cluster y seguir manteniendo la información que estaba manejando gracias a que los EJBs son distribuidos entre todos los servidores J2EE que formen parte de un Cluster.

Los Beans de Entidad se ejecutan sobre una única hebra y esto implica ciertos problemas de concurrencia que no existen en los Beans de Sesión, ya que múltiples clientes pueden acceder al Bean de forma simultánea.

Pero los EJBs de Entidad tienen un gran problema, como veremos con mas detalle en el siguiente apartado de implementación. Todas las cualidades que proporcionan tienen sentido cuando se maneja una pequeña, pero muy valiosa, cantidad de información, pero dejan de tener sentido en otro tipo de situaciones:

Cada EJB representa un registro en una Base de Datos. El ciclo de vida del EJB está completamente ligado al ciclo de vida de los datos que representa. Destruir un EJB es borrar el registro de la tabla y acometer el borrado (COMMIT). Crear un EJB es insertar un registro en la tabla y acometer la inserción. Para consultar un dato en la BBDD no hace falta crear un EJB sino simplemente solicitar una referencia a ese registro (lo que más adelante definiremos como Interfaces Remotas), es decir en cualquier caso se tienen que crear uno o varios objetos Java por cada registro de la tabla que se quiera manipular, añadir o borrar.

¿Qué ocurre si se quieren leer 10.000 registros de la base de datos a través de EJBs de Entidad?

Pues se crearían 10.000 referencias a los registros de la BBDD. Es decir, 10.000 objetos Java que "apuntan" a un registro de la BBDD cada uno.

¿Y si se quisiese añadir 10.000 registros?

Si se quisiese añadir 10.000 registros habría que crear 10.000 EJBs y ejecutar 10.000 INSERT independientes, con la consiguiente perdida de rendimiento y el gran consumo de memoria.

Este es el porqué se han descartado los EJBs de Entidad para este proyecto. Según los instructores de Sun Microsystems, los EJBs de Entidad son muy útiles en aplicación web de comercio electrónico o banca *"online"* donde el volumen de datos es relativamente pequeño y la integridad y seguridad de los datos ha de ser asegurada plenamente. En otro tipo de entornos (cargas/lecturas masivas de datos, necesidad de respuesta rápida etc.), como el que nos afecta en este proyecto, los EJBs de Entidad son completamente desaconsejables, por la lentitud y el gasto de memoria que suponen.

4. IMPLEMENTACIÓN DE EJBs

Un EJB se representa mediante 1 Clase y 2 Interfaces Java:

- *El Bean* propiamente dicho que esta representado por un Clase derivada de alguna otra, normalmente Object, y que implementa la Interfaz *SessionBean o EntityBean*, ya sea un bean de Sesión o de Estado.
- La *Interfaz Remota*, que implementa la interfaz *javax.ejb.EJBObject*
- La *Interfaz Local*, que implementa la Interfaz javax.ejb.*EJBHome*

La *Clase* debe definir todos los métodos que incluye la Interfaz SessionBean o EntityBean. Además debe implementar el método ejbCreate(params) con el número de parámetros necesarios para que los clientes puedan crear un Bean con una serie de parámetros iniciales. También se puede definir el método ejbCreate sin argumentos. En el caso de los Beans de Sesión sin Estado, este método no debe tener nunca parámetros pues no tendría sentido ya que los Stateless Session Beans no tienen atributos que inicializar. Por último es aquí donde se implementan los métodos de negocio necesarios para el EJB.

La *Interfaz Local* hace las veces de Factoría para la clase que implementa el Bean. Normalmente la Interfaz Local u Objeto Local se genera en el momento de la instalación del Bean. A través de estas factorías, los clientes pueden crear o destruir instancias de los Beans. Durante el proceso de instalación de la aplicación, las Factorías de EJBs se instalan en el espacio de nombres del servidor para que puedan ser localizadas por los clientes mediante consultas al árbol JNDI.

La *Interfaz Remota* es a través de la cual un cliente invocará los métodos de negocio del EJB. Por tanto debe definir los métodos de negocio necesarios. Esta interfaz hereda otros métodos definidos en su superclase *EJBObject*. Hace la veces de Proxy para el Objeto EJB real. Según la especificación de Sun es necesaria esta Interfaz Proxy para que el cliente no pueda invocar los métodos del EJB directamente y así se aseguren las posibilidades de ofrecer servicios transparentes como las transacciones. Esta Interfaz recibe las peticiones de los clientes cuando invocan los métodos de negocio, y delega su ejecución a una instancia de una Clase EJB.

Ejemplo de una Clase EJB : Clase StaffBean

```
package edu.ucm.sip.commons.DAO;
import javax.ejb.SessionContext;

public class StaffBean implements javax.ejb.SessionBean
{
    public void ejbCreate() {
        //Su EJBCreate no recibe argumentos por tanto es Stateless (Sin Estado)
    }

    //Metodo Auxiliar
    private Connection getConnection() {  //codigo dl metodo auxiliar  }

    // Metodo de Negocio
    public Collection getStaffMembers()
    {//Codigo del Metodo de Negocio
    //Accesos a la Capa de Datos ..etc
     }

    // Otro metodo de negocio sobrecargando el anterior
    public Collection getStaffMembers(ArrayList query) {//Codigo Metodo de negocio}

    /* Debido a que este es un Bean de Sesion sin Estado (Stateless), no necesitamos
     agregar ningún código especial para los siguientes metodos*/
    public void ejbRemove() {;}
    public void ejbActivate() {;}
    public void ejbPassivate() {;}
    public void setSessionContext(SessionContext sC) {;}
}//Fin de la Clase
```

Ejemplo de la Interfaz Local asociada al EJB anterior. Solo define el método de creación que devuelve una referencia a la Interfaz Remota:

```
package edu.ucm.sip.commons.DAO;

import java.rmi.RemoteException;
import javax.ejb.CreateException;

public interface StaffHome extends javax.ejb.EJBHome {

   public StaffRemote create() throws RemoteException, CreateException;

}
```

La Interfaz Remota, define los métodos de negocio del EJB, pues es a través de esta interfaz como un cliente invocará dichos métodos:

```
package edu.ucm.sip.commons.DAO;

import javax.ejb.EJBObject;
import java.util.Collection;
import java.util.ArrayList;
import java.rmi.RemoteException;

public interface StaffRemote extends EJBObject {

   public Collection getStaffMembers() throws RemoteException;

   public Collection getStaffMembers(ArrayList query) throws RemoteException;

}
```

5. INSTALACIÓN DE LOS EJBs EN EL SERVIDOR DE APLICACIONES

5.1 El descriptor de Despliegue (DD)

El descriptor de Despliegue (Deployment Descriptor o DD) es un fichero que contiene las propiedades de un EJB descritas con la sintaxis de XML incluyendo información estructural, información sobre el ensamblaje de la aplicación y lista de recursos necesarios.

El DD permite mantener las propiedades de los EJB fuera de su código. Esto permite al responsable de despliegue modificar los parámetros en el momento del despliegue o incluso "en caliente", si el servidor de aplicaciones lo permite.

El DD describe atributos, como nombre y valores de variables, nivel de control de transacciones para cada método, y listas de control de acceso para controlar el acceso a los beans y a sus métodos.

Se coloca normalmente en un fichero tipo jar dentro del subdirectorio META-INF, aunque es posible colocarlo junto con el resto de clases y paquetes siempre que esté bien especificado en el descriptor de despliegue su paquete. Su nombre debe ser ***ejb-jar.xml***. Un descriptor de despliegue puede describir múltiples beans.

5.1.1 Acceso a los parámetros del DD desde el Bean

Un Bean nunca accede directamente a los parámetros del DD. Dichos parámetros se cargan en el espacio de nombres del servidor y pueden ser localizados por el Bean a través del árbol de nombres y directorios JNDI.

5.1.2 Contenido del descriptor de despliegue

El descriptor de despliegue debe contener referencias a uno o más EJBs bajo la etiqueta <entreprise-beans>. Dentro de esta etiqueta se incluye una que indica que tipo de EJB es. En nuestro caso solo hemos usado Beans de Sesión y por eso la etiqueta <session id="nombreUnicoBeanSesion">.

Después, dentro de la etiqueta <session> o <entity> si es de Entidad, se incluyen otros parámetros del Bean:

- <display-name> El nombre para ser mostrado en herramientas de diseño y gestión de proyectos.
- <ejb-name> Su ejb-name o nombre en el árbol JNDI para ser localizado por los clientes.
- <home> Ruta de paquete de la Clase que implementa la interfaz Local
- <remote> Ruta de paquete de la Clase que implementa la Interfaz Remota
- <session-type> Si es Stateless o Stateful

- <transaction-type> Si se quiere que sea el Contenedor o el propio Bean quien gestione las transacciones con la base de datos.

-Ejemplo de ejb-jar.xml para el proyecto (Dos Beans de Sesión sin Estado) :

```
<?xml version="1.0"?>

<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>

    <enterprise-beans>

        <session id="staffEJB">
            <display-name>Staff Session Bean</display-name>
            <ejb-name>StaffEJB</ejb-name>
            <home>edu.ucm.sip.commons.DAO.StaffHome</home>
            <remote>edu.ucm.sip.commons.DAO.StaffRemote</remote>
            <ejb-class>edu.ucm.sip.commons.DAO.StaffBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>

        <session id="loginHistEJB">
            <display-name>Login Hist Session Bean</display-name>
            <ejb-name>LoginHistEJB</ejb-name>
            <home>edu.ucm.sip.commons.DAO.LoginHistHome</home>
            <remote>edu.ucm.sip.commons.DAO.LoginHistRemote</remote>
            <ejb-class>edu.ucm.sip.commons.DAO.LoginHistBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>

    </enterprise-beans>

</ejb-jar>
```

JBOSS permite añadir información no-estándar específica para su arquitectura y prestaciones. Además de configurar el fichero ejb-jar.xml, se puede añadir el fichero ***jboss.xml*** en el mismo directorio que el fichero ejb-jar.xml, en el que se puede especificar una serie de configuraciones de los EJBs relacionadas con prestaciones muy avanzadas de JBOSS que se salen del objetivo de este proyecto, no obstante era necesario mencionarlo por si en un futuro se desea investigar sobre esta serie de prestaciones adicionales.

Bibliografía

[GOF94]
Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (The Gang of Four). *Design Patterns*. Addison Wesley Professional Computing Series, 1994.
[PATTSUN]
http://java.sun.com/blueprints/corej2eepatterns/index.html
[PATTPC]
http://www.programacion.net/tutorial/patrones/

[JAKA03]
http://jakarta.apache.org/
[STRUTS]
http://jakarta.apache.org/struts/
[STXX1]
http://stxx.sourceforge.net/
[STXX2]
http://www.orbeon.com/oxf/doc/model2x-model2x
[STPC1]
http://www.programacion.com/tutorial/struts/
[STPC2]
http://www.programacion.net/articulo/tips_struts/
[MERC02]
Julien Mercay and Gilbert Bouzeid.
Boost Struts With XSLT and XML. Java World, Febrero 2002.

[WEBSA]
http://www.osmosislatina.com/aplicaciones/servidor_web.htm

Acceso Concurrente y Base de Datos

PROBLEMAS PARA GESTIONAR LA BASE DE DATOS CON ACCESO CONCURRENTE

1 de Mayo de 2004, v1.2

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

1. INTRODUCCIÓN

El origen de muchos problemas para gestionar los datos en aplicaciones web, es la principal característica de este tipo de aplicaciones: el acceso concurrente de varios usuarios a la misma.

Este documento pretende enunciar los problemas más comunes derivados del acceso concurrente, exponer algunas soluciones genéricas (primera parte), definir como afectan al proyecto estos problemas y comentar como evitarlos (segunda parte).

2. DESCRIPCIÓN DEL PROBLEMA

2.1 Conceptos Básicos

2.1.1 Conexión con Base de Datos

Una conexión es la vía de comunicación entre la aplicación y la base de datos. Una conexión sólo puede realizar una operación con base de datos a la vez; sin embargo, el gestor de base de datos (SGBD) puede dar servicio a varias conexiones simultáneas.

Normalmente la aplicación maneja una conexión por usuario cuando este va a realizar una consulta o modificar algún dato, aunque un proceso multi-hilo (multithread) puede necesitar varias conexiones.

Nuestra aplicación establecerá conexiones JDBC, este tipo de conexión son gestionadas por drivers JDBC propios del SGBD.

2.1.2 Bloqueo

Cuando dos o más conexiones están modificando el mismo dato, el SGBD puede bloquear todas las conexiones excepto una para realizar las operaciones, sin embargo no se puede garantizar qué resultado se obtendrá.

2.1.3 Transacción

Las transacciones (*commit – rollback*) permiten ejecutar bloques de código, incluidas sentencias SQL, sirven para encadenar varias sentencias en una misma transacción, es decir, podemos hacer varios INSERT y luego deshacer los cambios (rollback) de todos si no se ha cumplido alguna condición, para que los cambios tengan efecto sobre los datos hay que confirmarlos (commit). Por defecto, las conexiones JDBC se inicializan en modo autocommit, aunque esta opción se puede deshabilitar.

Las transacciones deben garantizar las propiedades **ACID (*Atomicity – Consistency – Isolation – Durability*).**

Atomicity: Varias operaciones encadenadas se consideran una sola, son instrucciones indivisibles, de forma que nada puede ejecutarse antes de terminar.

Consistency: Consistencia significa que solo van a registrarse datos válidos en la BBDD. Si por alguna razón, una transacción que sea ejecutada viola alguna de las reglas de consistencia de la base de datos, esa transaccion será deshecha (ROLLED BACK) y la base de datos será llevada de nuevo a un estado consistente con sus propias reglas. Por el contrario, si una transacción exitosa se ejecuta, ésta llevará a la base de datos de un estado consistente a otro igualmente consistente con sus reglas.

Isolation: El Aislamiento requiere que múltiples transacciones ocurriendo al mismo tiempo no interfieran unas con otras. Si dos o más transacciones son ejecutadas casi simultáneamente, éstas deben ejecutarse en un orden estricto, no se deben solapar para que los datos que una de ellas modifique no afecten al funcionamiento de la otra(s). Nótese que la propiedad de Aislamiento no asegura qué transacción se debe ejecutar antes si se da una situación similar, sino que sólo debe asegurar que las transacciones se ejecutan en un determinado orden.

Durability: Los efectos de las operaciones realizadas son permanentes.

2.2 Principales Problemas

2.2.1 Introducción

Se puede partir del hecho en que los problemas de base de datos en un sistema concurrente son los mismos o muy parecidos a los problemas en un sistema de archivos con un sistema operativo multitarea.

En principio, la propiedad **Isolation** garantiza que no se den problemas derivados por la concurrencia de usuarios, sin embargo, según el nivel de bloqueos de la base de datos pueden darse varios tipos problemas.

2.2.2 Dirty Reads

Se pueden dar malas lecturas, es decir, obtener datos incorrectos por culpa de transacciones que no han sido confirmadas. Una transacción lee datos actualizados por otra transacción, pero todavía no comprometidos.

Transacción 1	**Transacción 2**
Begin	
SELECT X /* X = 0*/	
X = X + 10 /* X = 10*/	
UPDATE X /* X=10 en BD sin confirmar */	*Begin*
	SELECT X /* X = 10 */
Rollback /* X = 0 en BD*/	X = X + 10 /* X = 20*/
End	UPDATE X /* X=20 en BD sin confirmar */
	Commit /* X = 20 en BD y comprometido */
	End

La transacción 2 debería haberse deshecho al igual que la 1 y advertir del error, o actualizar la X con 10 para mantener la coherencia, en cualquier caso 20 no es un valor correcto.

2.2.3 Non-repeateable reads

Es imposible obtener el mismo dato al leer el mismo registro varias veces. Una transacción relee un dato que ha cambiado sin haberlo modificado desde la primera lectura.

Transacción 1	**Transacción 2**

Bejín	
SELECT X /* X = 0*/	*Begin*
X no se modifica	X = 20
X no se modifica	UPDATE X /* X=20 en BD sin confirmar */
X no se modifica	Commit /* X = 20 en BD y comprometido */
SELECT X /* X = 20*/	*End*
End	

Para prevenir estas situaciones se pueden lanzar consultas bloqueantes (select for update) sin embargo, esto reduce la concurrencia de accesos a bases de dato y por lo tanto, disminuye el rendimiento. También se pueden realizar consultas preventivas antes de utilizar los datos, pero esto no ofrece garantía.

2.2.4 Phatom Reads

Una transacción relanza la misma query sucesivas veces y cada vez obtiene distinto número de resultados.

Transacción 1	**Transacción 2**
Bejín	
SELECT … WHERE cond /* 10 rows */	*Begin*
	INSERT /* 25 rows que cumplen cond */
	Commit /* insercciones comprometidas */
SELECT … WHERE cond /* 35 rows */	*End*
...	
End	

Esto rompe la propiedad de atomicidad, ya que la transacción 1 no debería ser interrumpida por ninguna otra, para evitar esto la transacción 1 debería bloquear los registros que cumplen *cond*, sin embargo, es poco recomendable permitir que las consultas bloqueen datos ya que afecta al rendimiento de forma considerable.

2.2.5 Conclusiones

En resumen se pueden encontrar dos tipos de problemas generales:

- Inconsistencia: Los datos disponibles han sido modificados antes de utilizarlos, estos ***Datos Falsos***, provocan inconsistencia en la base de datos.
- Incoherencia: Se tienen datos contradictorios sin haber sido modificados.

Como se puede ver en los ejemplos cada problema puede tener distintos tratamientos, cual es el más adecuado dependerá del contexto.

2.3 Soluciones Genéricas

Se pueden adoptar cuatro tipos de soluciones

2.3.1 Bloqueos Físicos

Niveles de aislamiento

Para evitar los problemas derivados del acceso concurrente, la primera medida que se puede tomar es eliminar el acceso concurrente a los datos.

Este tipo de bloqueos se denomina físico consiste en dejar que la primera transacción que accede a un determinado conjunto de datos termine y retener el resto hasta que la primera termine.

Esta técnica deteriora mucho el rendimiento de la aplicación ya que elimina la concurrencia al acceso de datos, lo que afecta a la eficiencia en la gestión de transacciones y además, requiere complejos algoritmos para implementarla.

Los drivers JDBC del SGBD son los encargados de realizar estos bloqueos, puesto que afectan tanto al rendimiento el nivel de aislamiento puede configurarse, cuanto más alto más bloqueos y menos concurrencia.

1) TRANSACTION_NONE: No se soportan transacciones (solo autocommit)
2) **TRANSACTION_READ_UNCOMMITED:** Pueden ocurrir "dirty reads", "non-repeatable reads" y "phantom reads"
3) TRANSACTION_READ_COMMITED: Pueden ocurrir "non-repeatable reads" y "phantom reads"
4) TRANSACTION_REPEATABLE_READ: Pueden ocurrir "phantom reads"
5) TRANSACTION_SERIALIZABLE: Elimina todos los problemas.

El segundo nivel es el más utilizado.

Consultas bloqueantes

También se pude aplicar esta técnica derivando la responsabilidad de los bloqueos al programador y no al SGBD, de forma que solo se realice un bloqueo físico cuando el programador considere que es necesario, para ello se utilizan SELECT FOR UPDATE, esta select bloquea todos los registros seleccionados hasta que se cumpla una condición. Esta forma es muy arriesgada, porque las transacciones sobre registros bloqueados quedan pendientes de que se liberen, lo que puede desencadenar en abrazos mortales sino se tiene cuidado y aún así se corre el riesgo de bloquear la base de datos si un proceso se detiene, provoca una excepción no controlada...

2.3.2 Soluciones Lógicas

Modificación del Modelo de Datos

La estructura de la aplicación puede estar orientada a mantener un control de las transacciones independiente del SGBD, esto se consigue manteniendo complejas estructuras de datos o normalmente modificando el modelo de datos.

Por ejemplo, se puede mantener una tabla con las columnas y filas de cada tabla que están siendo accedidas. Con esta información el programador puede bloquear y encolar transacciones, informar al usuario si esta mostrando información potencialmente falsa...

Utilizando la Lógica de Negocio

Conociendo las situaciones de riesgo, se pueden implementar mecanismos concretos para prevenirlos, capturar los errores, informar al usuario...

EJB

Dejar que el contenedor de EJBs gestione las transacciones a costa del rendimiento.

El programador sólo necesita especificar qué métodos son transaccionales, sin tener que modificar el modo autocommit de las conexiones, ni el nivel de aislamiento.

Además aunque en perjuicio del rendimiento, la tecnología EJB implementa transacciones distribuidas y automatiza la persistencia de objetos consiguiendo mayor portabilidad, potencia y sencillez. Esta solución es muy útil si no se desea tener un control fuerte sobre las transacciones pero requiere un servidor de aplicaciones que de soporte a esta transaccionalidad (JBOSS es completamente compatible) y un período de configuraciñon adicional. Puede ralentizar las aplicaciones si no se controla la transaccionalidad de algunas operaciones muy frecuentes.

2.3.3 No hacer NADA

Esta es con diferencia las solución más utilizada, normalmente se hace un estudio de la probabilidad en la que pueden producirse situaciones de riesgo y en el impacto que pueden tener, si resulta muy baja se toleran estas situaciones teniendo en cuenta que aunque se produzcan no tienen porque afectar a la calidad de los datos y si lo hacen el sistema se puede recuperar.

Un ejemplo claro de cuando una situación de riesgo puede obviarse es la siguiente, una transacción ha borrado todos los registros que otra esta modificando. Lo más fácil es capturar el posible error que lance el SGBD y anotarlo, ya que no importa que los datos hallan sido modificados o no si iban a ser eliminados de todas formas.

3. INTRANET

A continuación se realiza un análisis, organizado por funcionales, de cómo los problemas generales expuestos anteriormente pueden afectar a la Intranet con el fin de conocer qué problemas concretos hay que tratar y lo que es más importante en qué situaciones.

Después se tratará de generalizar estos problemas concretos para tratar de encontrar una solución general.

3.1 PERSONAL

Conflictos

- Un usuario esta modificando sus datos personales y mientras lo hace el administrador le da de baja.
    - Este efecto no es importante, puesto que el usuario va a ser eliminado con los datos modificados o no, sin embargo hay que controlar esta situación para que no cause un error en ejecución.

Dirty Reads

- Un usuario solicita un listado del personal, lo que origina que una transacción realice una consulta sobre todos los registros del personal. Inmediatamente después, un usuario esta modificando sus datos personales, lo que implica que otra transacción modifique los datos de un registro que ya ha sido leído por la primera y que aún no ha mostrado los datos.
    - Esto provoca que el listado no muestre los datos más actuales, por lo tanto, el usuario obtiene información falsa. Puesto que la información mostrada no es del todo importante y se puede corregir con un simple refresco, se puede obviar este riesgo.
- Un usuario modifica sus datos personales y mientras el mismo u otro con permisos suficientes también los modifica. Dos actualizaciones simultaneas del mismo campo son controladas por el SGBD, sin embargo, puede darse el caso de que un usuario vea sus datos y los cambie, pero antes de que el cambio se haga efectivo otro cambio se adelante.
    - Esto puede darse ya que varias personas físicas pueden ser el mismo usuario de la Intranet, ya que esta situación debería ser poco frecuente y el impacto no es grave, también se puede obviar el riesgo.

Non-Repeatable Reads

- El administrador esta modificando los datos de distintos usuarios, mientras varios usuarios están obteniendo un listado del personal. El resultado es que ninguno de los usuarios consigue el mismo listado a pesar de solicitarlo casi simultáneamente.
    - El impacto de esta situación es igual que en el primer caso y por lo tanto se podría ignorar.

Phantom Reads

- Un usuario solicita tres listados del personal simultáneamente y obtiene tres listas de distinto tamaño, porque el administrador esta dando altas y bajas de personal a la vez.
    - Esto es una situación anómala y fácil de detectar por los usuarios, por lo que tampoco resulta especialmente interesante controlar.

Resumen

Cualquiera de los casos expuestos anteriormente no resulta lo suficientemente importante como para deteriorar el rendimiento de la aplicación ya que no producen ningún

error en ejecución y son situaciones aisladas y fáciles de detectar por el usuario. Si bien es cierto que pueden resultar confusas y molestas si se dieran con cierta frecuencia.

4. POSIBLES SOLUCIONES

Implementación de la lógica de negocio mediante EJBs

Implementación de bloqueos lógicos (Véase Apéndice A)

5. Bibliografía

Apuntes de Integración de Sistemas de la Universidad de la Coruña
[http://www.tic.udc.es/~fbellas/teaching/is] (Tema 2, Apartado 2.1)

Especificación JDBC 3.0 de Sun
[http://java.sun.com/products/jdbc/download.html#corespec30]

Oracle Technology Network
[**http://otn.oracle.com/]**

Apéndice A

ORACLE y Concurrencia

Multiversioning

Oracle utiliza una técnica denominada 'multiversioning' (también llamada 'read-consistent') para evitar resultados inesperados en las sentencias. Esta técnica consiste en que la vista de la base de datos se mantiene igual a como estaba al principio de una sentencia.

Oracle aplica multiversioning a nivel de consulta, no a nivel de transacción. O sea, que si hacemos *SELECT count(*) FROM tabla,* y, mientras Oracle está contando filas, otra transacción hace un *INSERT* en la misma tabla y después *COMMIT*, este nuevo registro no será tenido en cuenta en el count. Sin embargo, si se vuelve a repetir el *SELECT count(*) FROM tabla*, obtendremos un registro más. Veámoslo detenidamente:

TIEMPO	TRANSACCIÓN 1	TRANSACCIÓN 2
1	SELECT count(*) FROM tabla	
2	Resultado = 8	
3	SELECT count(*) FROM tabla	
4		INSERT INTO tabla VALUES (1)
5		COMMIT
6	Resultado = 8	
7	SELECT count(*) FROM tabla	
8	Resultado = 9	
9	COMMIT	

Como se puede apreciar, el multiversioning no ocurre a nivel de transacción, o sea, que dentro de una misma transacción no tenemos la vista de la base de datos que teníamos al inicio de la transacción.

Locks

Un lock es un mecanismo para regular el acceso concurrente a un recurso compartido. Oracle pone locks a nivel de fila y los utiliza de la siguiente forma:

- Una sentencia **SELECT** nunca pone un lock
- Una sentencia **SELECT FOR UPDATE** pone un lock en las filas seleccionadas
- Una sentencia **DELETE** pone un lock en las filas borradas. Si se borran filas de una tabla padre y la tabla hija no tiene indexada la FK, Oracle pone un lock en todas las filas de la tabla hija

- Una sentencia **UPDATE** pone un lock en las filas actualizadas. ***Si se actualizan filas de una tabla padre y la tabla hija no tiene indexada la FK, Oracle pone un lock en todas las filas de la tabla hija***
- Una sentencia **INSERT** no pone ningún lock (aunque puede producir un bloqueo, como se explica en el apartado siguiente)

Como conclusión, Oracle bloquea lo mínimo imprescindible. El caso más peligroso es el UPDATE o DELETE de tablas padre con FK no indexadas en las hijas. Por esto, como norma: '**siempre hay que indexar las FKs**'. En el último apartado se incluye un script para ver FK no indexadas.

Blocking

Un bloqueo ocurre cuando una sesión quiere poner un lock en una fila que ya tiene un lock puesto previamente por otra sesión activa (si el lock perteneciera a una sesión no activa, se ignora el lock). En ese caso, la primera sesión tiene que esperar a que la sesión que puso el lock haga COMMIT o ROLLBACK.

Oracle utiliza los bloqueos de la siguiente forma:

- Una sentencia SELECT nunca se queda bloqueada
- Una sentencia SELECT FOR UPDATE se queda bloqueada si una de las filas seleccionadas tiene un lock (excepto en el caso en que se haga SELECT FOR UPDATE NOWAIT, en cuyo caso dará un error)
- Una sentencia UPDATE se queda bloqueada si una de las filas a actualizar tiene un lock
- Una sentencia DELETE se queda bloqueada si una de las filas a borrar tiene un lock
- Una sentencia INSERT se queda bloqueada si en otra transacción se ha hecho otro INSERT con la misma PK o con un valor igual que tiene restricción UNIQUE. El problema de la PK se puede resolver utilizando secuencias

Deadlocks

Un deadlock ocurre cuando una sesión A está bloqueada esperando a que se libere un lock que ha puesto una sesión B, y al mismo tiempo, la sesión B está esperando que se libere un lock que ha puesto la sesión A.

Oracle detecta los deadlocks, escoge aleatoriamente una de las sesiones que están bloqueadas y le da un error en la sentencia que le estaba bloqueando.

A continuación se exponen unas consultas que pueden ayudar a detectar este tipo de problemas en bases de datos en funcionamiento

Detectar FK no indexadas

```
select table_name, constraint_name,
       cname1 || nvl2(cname2,','||cname2,null) ||
       nvl2(cname3,','||cname3,null)  ||  nvl2(cname4,','||cname4,null)
||
       nvl2(cname5,','||cname5,null)  ||  nvl2(cname6,','||cname6,null)
||
       nvl2(cname7,','||cname7,null) || nvl2(cname8,','||cname8,null)
            columns
  from ( select b.table_name,
                b.constraint_name,
                max(decode( position, 1, column_name, null )) cname1,
                max(decode( position, 2, column_name, null )) cname2,
                max(decode( position, 3, column_name, null )) cname3,
                max(decode( position, 4, column_name, null )) cname4,
                max(decode( position, 5, column_name, null )) cname5,
                max(decode( position, 6, column_name, null )) cname6,
                max(decode( position, 7, column_name, null )) cname7,
                max(decode( position, 8, column_name, null )) cname8,
                count(*) col_cnt
           from (select substr(table_name,1,30) table_name,
                        substr(constraint_name,1,30) constraint_name,
                        substr(column_name,1,30) column_name,
                        position
                   from user_cons_columns ) a,
                user_constraints b
          where a.constraint_name = b.constraint_name
            and b.constraint_type = 'R'
          group by b.table_name, b.constraint_name
       ) cons
 where col_cnt > ALL
        ( select count(*)
            from user_ind_columns i
           where i.table_name = cons.table_name
             and i.column_name in (cname1, cname2, cname3, cname4,
                                   cname5, cname6, cname7, cname8 )
             and i.column_position <= cons.col_cnt
           group by i.index_name
        )
```

Detectar locks y bloqueos

```
select username,
       v$lock.sid,
           trunc(id1/power(2,16)) rbs,
           bitand(id1,to_number('ffff','xxxx'))+0 slot,
           id2 seq,
       lmode,
           request
from v$lock, v$session
where v$lock.type = 'TX'
  and v$lock.sid = v$session.sid
  and v$session.username = USER
```

OPTIMIZACIÓN DE BASES DE DATOS ORACLE

01 de Julio de 2004, v1.0

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

1. PRÓLOGO

Este documento resume de manera breve las diferentes técnicas que pueden ser aplicadas para optimizar el rendimiento de una base de datos Oracle.

Se va a concentrar la atención sobre la optimización que pueda ser efectuada por un desarrollador, desde el diseño del modelo de datos hasta la optimización de consultas. Oracle permite muchos más tipos de optimizaciones sobre el propio SGBD, pero ya se requerirían unos conocimientos muy avanzados correspondientes al curso Data Base Administrator (DBA) que ofrece Oracle para las diferentes versiones del producto.

2. INTRODUCCIÓN

Aplicando ciertas mejoras en la etapa de diseño del modelo de datos y optimizando cuidadosamente la forma en que se realizan consultas a dicha base de datos se puede mejorar enormemente el rendimiento, sin tener que modificar ningún parámetro del SGDM, para lo que harían falta permisos de administración sobre el SGDM y/o la posibilidad de modificar parámetros del SGBD que puedan afectar a todas las bases de datos residentes en dicho SGBD, cosas que no siempre son posibles en entornos de producción.

Todas las mejoras propuestas en este documento pueden ser efectuadas en la fase de diseño de la aplicación y en ningún caso deben afectar a otras bases de datos del mismo SGBD. Algunas de ellas, como el uso de planificadores, podrán ser modificadas "en caliente", sin tener que recompilar o redesplegar la aplicación, si la aplicación se diseña pensando en este detalle.

Como herramienta que puede ayudar a optimizar el rendimiento de consultas se puede destacar SQL Lab Tuning de Quest Software, que va incluido dentro de su herramienta mundialmente famosa TOAD for Databases en su versión de pago. **[TOAD]**

3. OPTIMIZACIÓN DEL MODELO DE DATOS

Esta optimización se refiere a algo mas concreto que diseñar un buen modelo de datos en el que se evite tener que hacer JOINS costosos sobre las tablas o que la información esté bien dividida y relacionada.

En el momento de creación de las tablas sobre el SGBD Oracle es posible indicar que se quieren tener en cuenta una serie de hechos que pueden optimizar el acceso a determinadas tablas muy consultadas o acelerar la consulta cruzada de información de varias tablas que se interrelacionan con mucha frecuencia.

Entre todas las posibilidades encontramos:

3.1 Índices

Los Índices son estructuras en forma de árbol que permiten acceso directo a determinadas filas de una tabla.

Se clasifican según varios criterios, entre los que destacamos los más relevantes según el propósito de este documento:

3.1.1 Índices Simples o Compuestos (Concatenados)

Si un índice es Simple sólo indexa una columna de la tabla. Un índice compuesto almacena información de varias columnas de una tabla. No es necesario que el índice sea creado en el mismo orden en que están creadas las columnas de la tabla, y tampoco es necesario que las columnas sean adyacentes.

3.1.2 Índices Únicos y No-Únicos

Los índices únicos garantizan que nunca va a haber dos valores iguales en la columna que esta indexada por ese índice. No se debe confundir con el concepto de Clave Primaria. Los índices no-únicos son aquellos en que para una clave pueden existir múltiples filas asociadas con él.

Como se ha dicho, el índice es una estructura de árbol-B en cuyas hojas hay punteros a filas de ciertas tablas indexadas. Con esto se acelera la búsqueda sobre los campos más consultados de las tablas principales de la base de datos. Son una herramienta muy potente ya que Oracle da mucha importancia a los índices. En situaciones en las que es imprescindible una búsqueda exhaustiva por todas las filas de una tabla localizar un registro, mediante la indexación de los campos en la estructura de árbol se consigue reducir dramáticamente el tiempo de respuesta de la consulta. Además las hojas del árbol-B que representa un índice, están doblemente enlazadas para favorecer las búsquedas en orden ascendente y descendente.

Se recomienda indexar todos los campos que aparezcan en las cláusulas WHERE de las consultas más habituales y además todas las Claves Extranjeras (Foreign Keys) de cada tabla.

Los índices pueden ser creados a la vez que se crean las tablas, o ser introducidos a medida que se estudia el rendimiento de la BBDD y las consultas más frecuentes.

Ejemplo:

CREATE UNIQUE INDEX intranet_user_login_name
ON INTRANET_USERS(LOGIN_NAME)

3.2 Mejoras en la creación de las tablas

3.2.1 Caché de Tablas

Al igual que ocurre con las sentencias SQL mas usadas, Oracle también puede almacenar en una especie de caché las tablas más consultadas.

En la sentencia de creación de una tabla podemos indicar mediante la cláusula CACHE que queremos que Oracle almacene en su Caché la información contenida en la tabla. Con esto conseguimos acelerar en gran medida las consultas sobre esta tabla, a costa de un gasto de memoria que puede llegar a ser muy grande dependiendo del tamaño de la tabla. Por ello, solo se recomienda guardar en la caché siempre (es decir desde la creación) las tablas "Maestras" de la BBDD, es decir, tablas pequeñas con datos fijos que son consultadas con mucha frecuencia y que son el núcleo de la información de la base de datos.

Normalmente, cuando se hacen búsquedas exhaustivas sobre una tabla, los datos cacheados son depositados en la sección Menos Recientemente Usada **(LRU)** del buffer de Caché. Esto significa que serán los primeros datos en ser eliminados de dicha memoria caché cuando se necesite espacio en el buffer. Si la tabla es cacheada los datos son depositados en la sección Mas Recientemente Usada del buffer de Caché (MRU), por lo que serán los últimos datos en ser eliminados en caso de ser necesario liberar espacio en el buffer.

Las tablas cacheadas pueden alterar el comportamiento del Optimizador Basado en Coste y no deben ser usadas sin un cuidadoso estudio previo. Son remendadas para tablas pequeñas muy frecuentemente consultadas y poco frecuentemente modificadas (Tablas Maestras). Ejemplo:

ALTER TABLE EMPLEADOS CACHE;

3.2.2 Tablas Organizadas por índices

Una tabla organizada por índices es como una tabla normal con uno o más índices en alguna de sus columnas, pero en vez de mantener dos segmentos separados para el índice (B-Árbol) y la tabla, el SGBD mantiene una única estructura que almacena el valor indexado, la clave primaria de la tabla, y los valores de las otras columnas de la misma fila de la tabla.

Dado que el propio árbol almacena los datos de la tabla, se requiere menos espacio que si se usasen tablas normales, ya que las estructuras de la tabla y el árbol de Índices se han fusionado en una sola estructura.

La Tablas Organizadas por Índices son adecuadas para accesos frecuentes a través de la clave primaria, que es imprescindible que exista para poder usar este tipo de optimización. Aceleran en gran medida los accesos mediante dicha clave cuando se solicitan búsquedas de concordancia exacta (matching) o búsquedas en rangos de valores.

```
CREATE TABLE nombreTabla
(
 columna1      tipo1,
columna2       tipo2
...
PRIMARY KEY (columna1,columna2)
)
ORGANIZATION INDEX nombre_Tablespace nombre_Índice
```

4. OPTIMIZACIÓN DE SENTENCIAS

La optimización del rendimiento de las sentencias SQL que se lancen sobre una base de datos Oracle puede hacerse atendiendo a diversas consideraciones, entre las cuales se puede destacar:

- Optimizar las propias consultas SQL en la fase de desarrollo y pruebas, eliminar redundancias, campos innecesarios... etc. Mejorar la forma de solicitar los datos en las SQL, evitando JOINS masivos, ordenando campos en la consulta, evitando ciertos tipos de cláusulas WHERE... etc.

- Mediante el uso de Planificadores de Oracle, HINTS (pistas) y GOALS (objetivos).

4.1 OPTIMIZACIÓN BÁSICA DE SENTENCIAS SQL

Simplemente reorganizando un poco la consulta SQL, sin modificarla en absoluto, podemos mejorar el tiempo de respuesta total del cliente que realiza la consulta. Algunos de los consejos a seguir son estos:

- En primer lugar no solicitar datos innecesarios en la cláusula SELECT pues puede ralentizar la búsqueda en diferentes medidas. Se han de revisar concienzudamente todas las consultas para reducir al mínimo necesario el número de campos solicitados.

- Alterar el orden de las tablas en la cláusula FROM (si es que hay mas de una tabla). Las tablas que aparezcan en dicha cláusula se deben ordenar de menor a mayor volumen de datos.

- En la cláusula WHERE se deberán indicar primero las uniones de las tablas (es decir tabla1.columna1 = tabla2.atributo7) y después los criterios de selección (LIKE, > ,<, = valor). Asimismo se debe modificar el orden de los campos de la unión de dichas tablas. En el lado izquierdo de cada unión se ponen los campos de la tabla con menor volumen de datos y en el lado derecho los campos de la tabla con mayor volumen de datos.

- En los criterios de selección del WHERE se deben indicar en primer lugar los que hagan referencia a los campos de la tabla con mayor volumen de datos (Tabla Directora), de este modo la "poda" de posibles resultados se hará de forma más rápida dado que afecta en primer lugar a la tabla con mas filas que leer. Después de los campos de la tabla directora, vendrán los de las demás tablas, ordenados de mayor a menor volumen de datos, por la misma razón que antes. En el caso de tener que ordenar las cláusulas relativas a una sola tabla, se deberán poner las condiciones en orden de mayor a menor tamaño de campo, dejando en último lugar los campos numéricos.

- Siempre que sea posible, liberar al gestor de base de datos de realizar ciertas conversiones de tipos y operaciones como pasar a mayúsculas, ordenar, sumar resultados... etc. Siempre será más rápido hacerlo mediante código.

- Usar IN en vez de una serie de AND y OR en las cláusulas WHERE cuando se esta realizando una búsqueda porque Oracle almacena los resultados de las evaluaciones temporales en memoria hasta que se han completado todas las evaluaciones , por lo que se aumenta enormemente el gasto de memoria y el tiempo de la consulta.

- Usar EXISTS en vez de IN siempre que se pueda. La función EXISTS busca la presencia de la primera ocurrencia de una fila que cumple el criterio seleccionado al contrario que la sentencia IN que busca todas las ocurrencias. Por ejemplo:

Dadas 2 tablas PRODUCTO con 1000 Registros y OBJETOS con 1000 Registros.

(A)

```
SELECT p.product_id
FROM   productos p
WHERE  p.item_no IN (SELECT i.item_no
                     FROM   objetos i);
```

(B)

```
SELECT p.product_id
FROM   productos p
```

```
WHERE  EXISTS (SELECT '1'
        FROM   objetos
        WHERE  i.item_no = p.item_no)
```

En la consulta A, todas las filas de las tablas ITEMS serán leídas por cada fila que haya en la tabla PRODUCTOS. El resultado serán 1,000,000 de lecturas en la tabla Objetos. En el caso de B, un máximo de una fila de la tabla OBJETOS será leída por cada fila de la tabla PRODUCTOS, reduciendo considerablemente la carga del procesador de esta sentencia y el tiempo final.

- Si hay alguna tarea que se repita muy frecuentemente y puede ser implementada mediante Procedimientos Almacenados o mediante TRIGGERS (Disparadores), codificados en PL/SQL (Programming Language SQL), deberían incluirse en el SGBD puesto que son, con diferencia, la forma más rápida de hacer operaciones sobre la BBDD, ya que son conjuntos de sentencias que ya han sido optimizados, calculados, y cacheados desde su creación.

4.2 PLANIFICADORES

Los Planificadores son la parte del SGBD Oracle que se encarga de analizar las consultas, probar todas sus diferentes variantes o *reescrituras* y decidir cuál es la forma más optima de lanzar cada consulta. Hay varios tipos de optimizadores, y cada versión de Oracle va incorporando mejoras sobre ellos o incluso nuevos optimizadores. Por introducir un poco este tema tan complejo se citan dos de éstos:

El ***Optimizador Basado en Reglas*** es el que Oracle utiliza por defecto en cada consulta. Toma su nombre de que Oracle tiene predefinidas una serie de reglas que dicen como puede ser ejecutada una consulta. Estas reglas son fijas y no tienen en cuenta tamaños ni tipos de tablas. Oracle analiza la consulta y determina cual de las reglas puede ser la que más beneficie el rendimiento de la consulta.

Algunas de estas reglas son:

- Acceder a una fila a través de su clave primaria o única (UNIQUE).
- Acceder a la tabla haciendo una búsqueda exhaustiva (FULL TABLE SCAN).
- Acceder usando un índice compuesto.
- Acceder usando un índice sencillo.

El ***Optmizador Basado en Coste*** en más complejo que el anterior porque tiene en cuenta mucha más información para decidir cómo va a ejecutar una consulta. Se podría decir que el optimizador anterior es *"ciego"* mientras éste es *"informado"*.

Para que éste optimizador sea efectivo necesita de una serie de datos estadísticos que se van a ir recopilando a medida que se usa la BBDD. Según esos datos el optimizador

produce una serie de reescrituras de la consulta aplicando diversas consideraciones mediante el uso de HINTS (pistas) y GOALS (objetivos).

Los GOALS son órdenes que se le dan al SGBD para que optimice las consultas para un objetivo prefijado. Así se le puede ordenar a Oracle que optimice la consulta para obtener lo antes posible las primeras filas de la consulta (FIRST_ROWS) . Algunos GOALS son, por ejemplo:

- ALL_ROWS, optimizar para obtener en el menor tiempo posible TODAS las filas del resultado de la consulta.

```
SELECT /*+ ALL_ROWS*/ nombre,direccion
FROM Usuarios
WHERE …
```

- FIRST_ROWS, optimizar para obtener lo antes posibles las primeras filas de la consulta aunque las demás filas tarden algo mas de tiempo.

```
SELECT /*+ FIRST_ROWS*/ nombre,direccion
FROM Usuarios
WHERE …
```

Las pistas o HINTS son consejos que se le dan a Oracle para que ejecute la consulta de una determinada manera. Oracle hará caso de estas pistas siempre que sea posible. Se le pueden indicar que Índices debe utilizar, que algoritmo de cruzado de tablas debe usar...etc. Ejemplo de HINTS:

```
SELECT /*+ MERGE_JOIN*/ nombre,direccion,sueldo
FROM Usuarios,Sueldos
WHERE …

SELECT /*+ INDEX(indice_nombre) */ nombre,direccion
FROM Usuarios
WHERE …
```

Una vez elaboradas las reescrituras aplicando los diversos HINTS, el Planificador Basado en Coste evalúa cada una de ellas en función del Coste respecto al impacto sobre los tres recursos principales del sistema : CPU, Memoria y Entrada/Salida. Entonces, basándose en estos costes y al objetivo (GOAL) que se haya fijado para la consulta, el planificador determina cual será la forma de ejecutar dicha consulta, es decir, cuál de las reescrituras deberá ser ejecutada.

Bibliografía

[OTN04]
http://otn.oracle.com/

[EFFSQL04]
http://www.oracle-base.com/articles/Misc/EfficientSQLStatements.php

[QUEST04]
Web de Quest Sotware. Fabricantes del SQL Lab Tuning y de TOAD. Programas que incluyen una amplia documentación sobre optimización de sentencias SQL
http://www.quest.com/

[TIPS04]
http://www.praetoriate.com/oracle_tips_sql_optimization.htm

[JM04]
http://www.lawebdejm.com/prog/oracle/

[REDC04]
http://www.redcientifica.com/oracle/

[TOAD]
http://www.quest.com/toad

Gestión de Errores

Intranet para un Dpto. Universitario

Gestión y manejo de excepciones

25 de 06 de 2004, v 2.0

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

1. PRÓLOGO

Este documento esta dirigido a los programadores que continúen la implementación del proyecto o que pretendan reutilizar el framework.

Se ha escrito con la idea de ayudar en la implementación, por lo que su lectura esta recomendada antes de empezar la tarea de codificación y no tiene especial interés para comprender el proyecto.

2. INTRODUCCIÓN

A lo largo del documento se explica como se realiza la gestión de errores, en que se ha fundamentado con el objeto de ofrecer una forma fácil de mantener un sistema de excepciones uniforme y estable.

Puesto que, es de vital importancia a la hora de ampliar la aplicación, por lo tanto, procuraremos explicar bien las medidas adoptadas sobre este tema.

El documento se ha estructurado en cinco partes:

- Introducción a la Gestión de Errores
- Tipos de Errores
- Gestión de Errores en la capa del Controlador
- Gestión de Errores en la capa del Modelo
- Gestión de Errores en la capa de la Vista

3. Introducción a la Gestión de Errores

Con la gestión de errores se persiguen tres objetivos fundamentales:

1. Retornar la aplicación a un estado después de producirse un error.
2. Ofrecer mensajes claros y correctos al usuario, generar trazas completas y que de verdad informen del error ocurrido y sus causas.
3. No alertar usuario de forma innecesaria y en caso de hacerlo ofrecerle una alternativa viable.

La gestión de errores siempre ha sido un tema conflictivo por varios motivos:

- Nunca se le da la importancia que merece.
- Cada programador trata los errores de una manera.
- Involucra todas las etapas del desarrollo.
- Afecta a todas las capas de la aplicación.

Los framework utilizados no implementan un mecanismo completo para gestionar errores, ni tampoco marcan las pautas de trabajo para tratar los errores. Por ello se ha decidió ampliar el tratamiento y gestión de errores que realiza Struts y StrutsCX, adaptándolo a las necesidades particulares del proyecto.

4. Tipos de Errores

Podemos distinguir tres tipos de errores:

- Sistemas: Errores producidos porque algún recurso ha dejado de funcionar correctamente:
  - Perdida de conexión con base de datos.
  - Servidor de aplicaciones fuera de servicio
  - Timeout por retardos en la red...
- Usuario: Son errores producidos por datos defectuosos introducidos por el usuario:
  - Incorrectos
  - Incoherentes.
  - Mal formateados...
- Programación: Son los errores producidos por una mala implementación.
  - Errores de diseño.
  - Errores en algoritmos
  - Errores de control...

Cada tipo merece un tratamiento distinto y no todos pueden ser tratados en las mismas capas de la aplicación, por ello hemos dividido la gestión de errores según las capas de la aplicación.

5. Gestión de Errores en la Capa del Controlador

El tratamiento de errores que se realiza en controlador afecta a dos partes del código:

Acciones

Las acciones deben capturar adecuadamente las excepciones que lanza la capa del modelo, siguiendo los consejos explicados en el documento de excepciones.

Las excepciones procedentes de la capa del modelo, pueden ser errores recuperables o al menos personalizables, por ejemplo:

- Se recibe una UserNotFoundException, la acción puede optar por presentar un mensaje al usuario para que introduzca de nuevo los datos del usuario entendiendo que los ha introducido los datos de un usuario que no existe.
- Se recibe un SQLException, se puede interpretar como un error de programación. De forma que se loga el error para poder ser depurado y se advierte al usuario dejándole la posibilidad de volver a intentarlo.
- No se consigue establecer conexión con base de datos, no se encuentra el datasource… se puede optar por intentar el acceso un par de veces, sin advertir al usuario y si el error persiste dirigir al usuario a una página de error final.

Además una acción puede provocar excepciones por si misma, normalmente estos errores son irrecuperables. Pueden ser de dos tipos:

- De programación, puesto que es la última capa que enlaza con el usuario no se puede implementar lógica alguna que recupere el estado.
- De sistema, faltan parámetros en la sesión (idioma, encoding…) ante esta circunstancia si se consigue identificar el error se puede optar por utilizar un valor por defecto, sin embargo, esta decisión es peligrosa ya que puede producir resultados falsos, en la mayoría de los casos sería necesario advertir al usuario o pedirle más información, no obstante suele ser muy difícil identificar el error exacto y lo habitual será cerrar la sesión y dirigir el flujo hacia la página de error.

Descriptores

La configuración de los descriptores juega una baza fundamental en el control de errores. En el descriptor de la aplicación (web.xml) se puede configurar la página a la que debe redireccionar el servidor de aplicaciones en caso de detectarse un código de error determinado o alguna excepción en particular.

Se ha propuesto presentar la página por defecto del servidor de aplicaciones cuando se encuentre con códigos de error, del tipo 400, 404, 500… ya que normalmente exceden al control del programador y poco puede hacer este por corregir el error.

Por otro lado, se propone que las acciones sólo puedan lanzar ServletException, de esta forma se puede conducir al usuario a una página de error amigable que muestre toda la información.

Para que este sistema funcione las acciones deben actuar como filtro de excepciones, capturando todas las excepciones que se puedan lanzar e intentando subsanar cada error, en caso de que no se pueda habrá que envolverlas como una ServletException y relanzarlas. Hay que tener especial precaución con las RuntimeException, en especial las provocadas por la capa del modelo, sino se capturan puede conducir el sistema a un estado inestable e impredecible. Por otro lado, en caso de capturar una excepción no comprobada (Runtime) es casi imposible identificar el error, por ejemplo: NullPointerException con lo cual no sólo no se puede recuperar un estado estable, sino que tampoco se puede informar al usuario o desarrollador claramente de la causa del error, por lo que se debe logar la traza completa de la excepción y terminar la sesión de la forma más elegante posible.

El web.xml podría configurarse del siguiente modo, utilizando las etiquetas definidas en la especificación J2EE **[J2EE]**.

Para tratar los *internal errors* del servidor.

```
<error-page>
    <error-code>500</error-code>
    <location>/web/pages/errorpage.jsp</location>
</error-page>
```

Para tratar las excepciones lanzadas desde las acciones.

```
<error-page>
    <exception-type>javax.servlet.ServletException</exception-type>
    <location>/testError.do</location>
</error-page>
```

6. Gestión de Errores en la Capa del Modelo

La gestión de errores que debe efectuarse en el modelo, es más complicada de explicar que de implementar.

Por un lado, durante la implementación del modelo la gestión de errores es hasta cierto punto flexible, siempre y cuando se respeten las normas especificadas en el documento de excepciones. Sin embargo, tener un contenedor de EJB de por medio cambia bastante el tratamiento de excepciones.

El contenedor de Ejes (JBOSS) esta regido por la especificación J2EE que tiene un apartado exclusivo para el tratamiento de excepciones. No es el objeto de este documento explicar cada una de estas especificaciones **[SUNWEB1]** nos limitaremos a comentar las que afectan al proyecto y explicar como se ha tratado.

Una acción no trabaja directamente con los objetos EJB sino que lo hace a través de un interfaz y es el servidor el que se encarga de gestionarlos.

El servidor puede lanzar dos tipos de excepciones **[EJB11]**:

- **Excepciones de Aplicación**: Reservadas para errores provocadas por la lógica de negocio
  - *javax.ejb.CreateException*
  - *javax.ejb.RemoveException*
  - *javax.ejb.FinderException*

- **Excepciones de Sistema**: Reservadas para errores provocados por fallos del sistema.
  - *javax.ejb.EJBException*

Por un lado, se ha de diseñar los DAO, VO… (EJB) como cualquier otra clase, estudiando que errores pueden provocar, analizar como tratarlos y como exponerlos en forma de excepciones.

Y por otro lado, hemos de comunicar el modelo con la acción (controlador) a través del contenedor (JBOSS).

Para ello se han tomado las siguientes decisiones que deben ser respetadas para que el control de errores sea estable y eficaz, basándonos en el capitulo *Exception Handling* **[EJB11]**:

- Los DAO lanzaran uno de los tres tipos de excepciones de aplicación que más se adapte al error detectado, para ello tendrán que capturar y envolver las distintas excepciones que lance el sistema o la propia lógica de negocio implementada en otros módulos. Añadiendo a la excepción información suficiente para que la acción pueda actuar consecuentemente.
- En caso de producirse errores del sistema se lanzarán RuntimeException en forma de EJBException, siempre que impliquen un error irrecuperable o como la excepción original si se piensa que la acción podrá capturarla y tratarla adecuadamente.

7. Gestión de Errores en la Capa de la Vista

La vista debe desempeñarse dos labores fundamentales a la hora de gestionar errores:

Gestionar errores del usuario

En numerosos puntos de la aplicación el usuario puede introducir datos, estos datos pueden ser "*malos*" por varios motivos:

- *Incompletos*: Si no se suministran los suficientes datos, normalmente la propia acción es capaz de detectarlo antes de mandar la información al modelo y puede tomar las medidas adecuadas como pedir los datos de nuevo.
- *Incoherentes*: Detectar datos incoherentes suele requerir chequear abundante casuística y por eso suele tratarse en la lógica del negocio (modelo).
- *Mal formateado*: Introducir fechas, teléfonos, e-mails… mal formateados puede detectase fácilmente mediante la utilización de expresiones regulares, de forma que la acción puede comprobarlas y actuar de forma consecuente.

Para tratar los errores que puede manejar la acción se utiliza la herramienta Validator **[JAKVAL01]** de apache. Esta herramienta se configura mediante dos ficheros XML (Ver documento sobre la capa del controlador).

En el *validator-rules.xml* se definen las reglas que puede chequear el framework. Por ejemplo: longitud mínima, expresiones regulares (mask)…

En el *validator.xml* se especifican que reglas que deben aplicarse a cada campo de cada formulario. Por ejemplo: La longitud mínima del campo password del formulario de login debe ser de 5 caracteres.

Esta herramienta ofrece la posibilidad de realizar la comprobación de reglas dinámicamente.

La mayor ventaja que aporta la utilización de esta herramienta es que se genera un XML con la información necesaria para que desde un XSL se pueda mostrar fácilmente el mensaje de error.

Mostrar errores

La vista debe ser capaz de mostrar información sobre los errores producidos, para ello se utilizan dos métodos:

Un JSP para mostrar errores irrecuperables. Este JSP esta declarado cómo página de error **[JSPPC01]** y se puede acceder a el a través de una redirección automática por parte del servidor de aplicaciones o manualmente desde alguna acción.

El otro método se reserva para expresar errores en los datos introducidos mediante un formulario. Consiste en preparar los XSL para detectar mensajes en los XML fuentes y con dicha información añadir al lado de cada campo (en rojo) la descripción del error.

Para facilitar la labor de los XSL se ha extendido la funcionalidad del DocumentBuilder de StrutsCX **[STCX1]** para incluir información adicional proporcionada por la excepción o la propia acción.

Bibliografía

[J2EE]
http://java.sun.com/j2ee
[EJB11]
Enterprise JavaBeans™ Specification, v1.1

[PATTSUN]
http://java.sun.com/blueprints/corej2eepatterns/index.html

[SUNWEB1]
Desarrollo de Componentes Web con la Tecnología JavaTM (SL-314)
Manual del Alumno (Sun Microsystems, Inc)
[JSPPC01]
http://www.programacion.net/tutorial/servlets_jsp/

[JAKA03]

http://jakarta.apache.org/

[JAKVAL01]
http://jakarta.apache.org/struts/api/org/apache/struts/validator/package-summary.html
[JAKVAL02]
http://wiki.apache.org/geronimo/Architecture/Validator

[STCX1]
http://it.cappuccinonet.com/strutscx/index.php

[WEBSA]

http://www.osmosislatina.com/aplicaciones/servidor_web.htm

CONTROL DE VERSIONES

Este documento no esta cerrado y se irá ampliando según sea necesario hasta finalizar el proyecto.

1.0 → Control de errores general.

1.1 → Control de Excepciones en EJBs.

2.0 → Apéndice de Excepciones pasa a ser un documento independiente.

Excepciones

Intranet para un Dpto. Universitario

Gestión y Manejo de Excepciones

25 de Junio de 2004, v 1.0

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

1. PRÓLOGO

Este documento esta orientados a programadores que no tengan claro como manejar los sistemas de excepciones de los modernos lenguajes de programación y para programadores experimentados en el uso de estos sistemas que no sepan resolver los problemas que implican su utilización.

2. INTRODUCCIÓN

Un refinado sistema de excepciones es una de las principales ventajas que ofrecen los lenguajes de programación modernos. Sin embargo, muchos programadores experimentados aún no saben como utilizarlas correctamente.

El objetivo de este documento es presentar y tratar de aclarar los problemas que conlleva el uso de excepciones en un lenguaje moderno como JAVA. No se pretender explicar la sintaxis de las excepciones sino profundizar en su comportamiento para llegar a conclusiones sobre su correcta utilización, por lo tanto, el texto está dirigido a programadores familiarizados con el tema, que sepan construir y lanzar excepciones (throw/s), como funcionan los bloques (try..catch..finally)... No obstante, a lo largo del documento se refrescaran algunos conceptos fundamentales.

3. Origen de las Excepciones

Cuando un programa incumple algunas de las restricciones semánticas del lenguaje, algunos lenguajes, se limitan a detener la ejecución, otros informan del error y continúan o simplemente se comportan de una forma impredecible.

En el caso de JAVA, la maquina virtual de java (JVM) indica el error al programa mediante una excepción; de forma que, cuando se produce un error se lanza una excepción y se captura donde puede ser gestionado dicho error.

Con este comportamiento se aporta robustez al lenguaje, ya que un error no tiene porque implicar la finalización del programa, se puede informar del error concreto y / o tratar adecuadamente donde se pueda, también se gana portabilidad al mantener una documentación implícita sobre los errores que se pueden producir.

En general, un sistema de excepciones también ofrece una alternativa a la utilización de los valores de retorno para indicar que se ha producido un error. Esto implica mantener un código de error asociado a cada fallo y según la función es fácil que estos códigos entren en conflicto con valores correctos de las funciones. Además la experiencia ha demostrado que los valores de retorno no suelen ser comprobados por los invocadores.

Una excepción puede ser lanzada por tres razones:

1.- Se produce un error inesperado y no previsto en una parte del programa cuando se esta ejecutando otra. Hablamos de excepciones asíncronas:

- Un error interno de la JVM
- Se invoca el método stop de Thread.

2.- Si la JVM detecta la ejecución anormal de una condición. Se habla entonces de excepciones **sincronizadas**, por ejemplo:

- Al evaluar una expresión viola las reglas semánticas de la JVM (división entre cero).
- Se excede alguna limitación SW/HW (disco lleno).
- Se pretende acceder a recursos no definidos (puntero a null)
- Error al linkar un programa.

3.- Se lanza a propósito, mediante la ejecución de la sentencia *throw*.

Cada una de las anteriores razones, puede tener su origen en tres hechos:

1.- Errores en la implementación del código.

2.- Error causados por datos incorrectos introducidos por el usuario.

3.- Cuando un recurso físico excede sus limitaciones o produce un error.

Aunque se hace referencia al lenguaje JAVA lo anteriormente citado aplica a cualquier lenguaje que incorpore un sistema de excepciones.

4. Conceptos Básicos

4.1 Throw / Throws / Wrapper

Un programador que esta diseñando una función puede decidir lanzar / disparar una excepción si por ejemplo, ha recibido parámetros de entrada incorrectos. Para lanzar la excepción se utiliza la palabra reservada ***throw***.

Continuando con el ejemplo, el programador que recibe la excepción porque los parámetros son erróneos, puede tratar de corregir los parámetros erróneos, escogiendo valores por defecto..., sin embargo, puede no tener información suficiente para reparar el error, es entonces cuando decide relanzar la excepción a quien ha invocado la función para que éste la trate, a su vez puede ser relanzada una y otra vez. Para relanzar excepciones se utiliza ***throws***.

Una función que contiene código que puede lanzar excepciones, puede decidir relanzarlas o capturarlas y manejarlas. Una variante de relanzar excepciones es combinar estas dos alternativas, capturando la excepción original y lanzando una nueva creada a partir de la original, es decir, envolverlas (*wrapper*). Se puede implementar de dos formas creando un nuevo tipo de excepción basada en la recibida que aporte más información o incluyendo información extra en la excepción para facilitar el posterior tratamiento del error. Puede ser muy útil, si por ejemplo, en un punto del flujo no tenemos información suficiente para corregir el error pero si podemos aportar algo a niveles superiores.

4.2 Catch y Finally

Se dice que una excepción se lanza (throw) en el punto donde ocurre un error y se transfiere el control (throws) al punto donde es capturada (***catch***) con el objetivo de poder tratarla adecuadamente.

Cuando una función detecta un error y lanza una excepción, detiene su ejecución, no obstante es muy probable que necesite terminar alguna acción, por ejemplo, liberar recursos hardware, para ello se utilizan los bloques ***finally***.

4.3 Checked y Uncheked

Al margen del lenguaje de programación, se distinguen dos tipos de excepciones comprobadas (*Checked*) y las no comprobadas (*Unchecked*).

El compilador comprueba que las checked exception son capturadas o relanzadas por todo el flujo después de su posible lanzamiento. Esto obliga a las funciones que invocan a funciones que pueden lanzar excepciones comprobadas a relanzarlas a su vez o a capturarlas y manejarlas.

Las excepciones no comprobadas no son tratadas en tiempo de compilación, si se dispara alguna excepción no comprobada y no es capturada el sistema se hará cargo de manejarla deteniendo la ejecución del programa.

Ambos tipos de excepciones pueden ser declaradas y creadas por los programadores, aunque solo esta obligado a declarar las comprobadas.

5. Utilización de Excepciones

Resulta realmente complicado explicar la forma correcta de utilizar las excepciones, ya que por su naturaleza son muy flexibles y se pueden aplicar de muchas formas según la situación.

En cualquier caso, para utilizar correctamente las excepciones, lo primero que hay que tener claro es saber para que se sirven, es decir, establecer los objetivos que se persiguen con la incorporación de excepciones en el código.

Objetivo Principal

De aquí en adelante, es importante que se tenga constantemente en cuenta lo siguiente:

- **Lanzar y capturar excepciones tiene el único fin de volver a un estado seguro donde**:

- Si es posible, permitir al usuario ejecutar otros comandos.
- Sino, guardar el estado automáticamente, para no perder información.
- En el peor de los casos, terminar la ejecución de una forma agradable.

Independientemente de lo que se mencione en los siguientes apartados se deberá mantener una Visión Global sobre este objetivo.

Objetivo Secundario

Antes de utilizar un sistema de excepciones hay que preguntarse ¿qué quiero que hagan por mi? La respuesta a esta pregunta no es sencilla, ni única y sus diferentes respuestas nos marcaran objetivos secundarios.

- Quiero que los mensajes al usuario final sea claros, concisos y comprensibles por él.
- Quiero que mi código sea comprensible para otros programadores independientemente de los errores que se puedan producirse.
- Quiero que mi código sea reutilizable en otros módulos.
- Quiero que mi código sea portable a otros sistemas / lenguajes.
- Quiero que mi código sea independiente de los recurso que utiliza.
- ...

Los dos problemas fundamentales a la hora de trabajar con excepciones son:

- Qué hacer cuando se produce o detectamos un error.
- Qué hacer cuando recibimos notificación de una excepción.

- Qué hacer cuando se produce o detectamos un error

Podemos lanzar una excepción para informar a un nivel superior del error o tratar el error y continuar.

Ninguna de las dos alternativas es mejor, dependen del contexto y de hecho no siempre podemos optar entre las dos. En general, es mejor tratar de solucionarlo y sino se puede informar del error. Al igual que se hace en la mayoría de los protocolos de comunicaciones para establecer conexión.

- Qué hacer cuando recibimos notificación de una excepción

Existen tres alternativas:

Captura la excepciones para informar del error de alguna forma(log...) e intentar corregir el fallo o conducir al sistema a un estado seguro.

Envolver la excepción, añadiendo la información útil de la que se disponga.

Relanzar la excepción directamente.

Normalmente, es recomendable la primera alternativa, pero como en el punto anterior no siempre es posible.

Para saber como actuar hay que hacerse estas preguntas:

- ¿Podemos reparar el error? Si la respuesta es afirmativa debemos capturar la excepción y corregir el estado.

En caso contrario:

- ¿Podemos añadir más información al error? Si la respuesta es afirmativa, debemos añadir la información de la que dispongamos y relanzar la excepción.

En caso contrario:

- ¿La excepción puede tener sentido más arriba? Si no es así hay que envolver la excepción, expresando un error más abstracto.

Si no se sabe tratar una excepción y tampoco se puede saber si módulos superiores sabrán tratarla o lo harán, lo mejor es reportar el error y relanzarla.

En cualquier caso, no existe un criterio claro de cómo utilizarlas, lo único que se puede asegurar con certeza es como **no** se deben utilizar. A continuación enumeramos una serie de consejos generales y algunas normas de cómo actuar en situaciones concretas, estás clasificados en tres puntos.

- Uso
- Creación
- Captura
- Lanzamiento

1. Uso

- Comprobar las condiciones conflictivas para saber si pueden fallar, antes de provocar una excepción.

Por ejemplo, obsérvese el siguiente fragmento de código:

```
if (!pila.empty()) {
    cima = pila.pop();
} else // código alternativo
```

Se puede pensar que comprobar si la pila esta vacía antes de extraer un elemento es una perdida de rendimiento, cuando pocas veces va a estar vacía, de forma que podemos sustituir lo por el siguiente fragmento.

```
Try {
    Pila.pop
} catch(EmptyStatckException ese) {
    // código alternativo
}
```

Esta decisión es un claro **error**, en este caso, hacer la comprobación es del orden 220 veces más rápido que lanzar y capturar la excepción, es decir, manejar una sola excepción (un desapila incorrecto) supone el equivalente a hacer 220 comprobaciones. Además se consigue romper el flujo del programa, complicar el código y al hacerlo dependiente de una clase ajena aumenta el acoplamiento.

- Utiliza las excepciones sólo para casos excepcionales, nunca para controlar el flujo.

- No hay que escatimar el uso de excepciones, utilizar tantos tipos de excepciones como se necesiten. Sin embargo, es mejor utilizar pocas que abusar. Para saber si se necesita una nueva excepción se debe pensar en el uso que se hará de ellas.

- No hay que describir la implementación sino el concepto del error, es decir, proporcionar mensajes como el “El fichero no puede abrirse porque no tiene permisos de lectura” en lugar de “Acceso prohibido al descriptor del fichero para el FileStringBuffer”.

- Utiliza excepciones comprobadas para casos recuperables y no comprobadas para errores de programación o para informar de precondiciones no cumplidas que imposibilitan continuar con la operación.

2. Creación

- Antes de crear una excepción nueva hay que asegurarse que no existe ninguna con el mismo significado en el sistema.
- Tener muchos tipos de excepciones permite recoger muchos errores particulares. Además si están agrupadas en una jerarquía resulta muy cómodo para manejarlas como una.
- No hay que complicar en exceso las jerarquías de excepciones, suele ser mejor parametrizar algún constructor. En general, tener más de cuatro o cinco subclases de una única excepción, indica que el diseño esta mal.
- Tanto al capturar como al lanzar hay que dar mensajes apropiados, claros y conciso, las excepciones personalizadas facilitan esta labor al programador.
- Un buen ejemplo de excepciones parametrizables son la SQLException.
- Crear una excepción comprobada o no depende mucho del uso que se le quiera dar:
    - Si se pretenden evitar valores de retorno complicados, es mejor utilizar comprobadas ya que se declaran y es obligatorio que se capturen o relancen.
    - Si se quiere informar de un error, es recomendable utilizar uncheked ya que nadie debe preocuparse de capturarlas o lanzarlas, lo que no quiere decir que no sean adecuadamente capturadas por quien deba hacerlo.

3. Capturar

- Si se capturan todas las clases de excepciones en un mismo bloque *catch* no es intuitivo hacer distinciones entre ellas, lo que complica generar mensajes de error concretos... por otro lado, es muy probable que la única acción que realizamos para todas las clases de excepciones sea reportar el error al log y mandar al usuario al mismo estado seguro, por lo tanto, no es necesario un bloque catch por cada excepción. En definitiva, un catch general solo tiene sentido si todas las excepciones se tratan igual, en tal caso, hay que preguntarse si es correcto lanzar tantas excepciones distintas.

- Capturar las excepciones lo más cerca que sea posible del código que las origina, ya que si bien es cierto que según ascendemos en la cadena es más fácil interpretar y corregir el error, hay un punto a partir del cual se pierde la trazabilidad.

- Siempre que sepamos tratar una excepción, debemos manejarla, no dejar que se capture en otro nivel superior. No hay que dejar margen al azar, si pasa algo no previsto se debe impedir que afecte al usuario final. En cambio, si no se puede manejar una excepción correctamente, no hay que capturarla.

- Hay que capturar solo aquello que se lanza, si capturamos Exception por ahorrar un catch corremos el riesgo de ocultar alguna RuntimeException... Además capturar solo lo necesario es una forma de documentar el código implícitamente.

No se deben ocultar excepciones. Es cierto, que puede ser molesto tener que declarar una excepción en la lista de throws de toda la cadena de métodos para que probablemente jamás se produzca la excepción por eso un programador poco experimentado esta tentado a hacer cosas como esta:

```
Try {
   // código que lanza excepciones no declaradas
}catch(Exception e) {
}
```

Con esto sólo conseguimos ignorar todas las excepciones las que queremos ignorar y el resto, de forma que si el código conflictivo lanza una RuntimeException, el programa falle en cualquier otro punto y sea muy difícil depurar el error.

- Si en un bloque de código, existe código inmediatamente después de un catch, cabe preguntarse si las excepciones capturadas son realmente excepciones o desviaciones del flujo básico del programa.

- Cuando se captura una excepción jamás hay que ocultarla, ni siquiera las típicas excepciones que se capturan dentro de un bloque finally. Cómo mínimo hay que reportarlas a un fichero de log, para ayudar a encontrar errores de programación.

- No hay porque capturar todas las excepciones que se lanzan, solo aquellas que se puedan tratar.

- Informa de la excepción en el momento de capturarla a menos que se tenga planeado relanzarla.

- No hay que gestionar las excepciones por separado.

Veamos otro ejemplo:

```
For (i=0;i<100;i++) {
   Try {
         N = pila.pop();
   } catch(EmptyStatckException ese) {
         // error: la pila esta vacía
   }

   Try {
         Out.write(n);
   } catch(IOException ioe) {
         // error: No se puede escribir en el fichero
   }
}
```

De esta forma se complica excesivamente el código sin conseguir nada, si sabemos como actuar en caso de encontrar la pila vacía, deberíamos preguntar primero y actuar (if ... else) si por el contrario no podemos hacer nada más que informar del error, no tiene sentido que sigamos intentado desapilar y escribir, ya que el error va a persistir.

```
Try {
   For (i=0;i<100;i++) {
         N = pila.pop();
      Out.write(n);
   }
} catch(EmptyStatckException ese) {
   // error: la pila esta vacía
} catch(IOException ioe) {
   // error: No se puede escribir en el fichero
}
```

4. Lanzamiento

Si la decisión de cómo y donde capturar una excepción puede resultar complicada, las decisiones que hay que tomar a la hora de lanzarlas son, cuando menos, igual de conflictivas.

Para ayudar a tomar estas decisiones podemos enumerar algunas reglas:

- No hay porque descartar automáticamente la propagación de excepciones, si bien es cierto, que se debe evitar si es posible manejar la excepción en un punto. Propagar las excepciones no es síntoma de baja calidad, ni de mala programación, ya que es igual de peligroso ocultar excepciones como relanzarlas sin control, solo hay que tener cuidado en no propagarlas por defecto.

- En especial cuando se programa siguiendo una metodología dirigida por contrato (DbC). Aún más importante que documentar el funcionamiento de un módulo, es documentar las excepciones que puede lanzar:
  - Que significan.
  - Que información aportan.
  - Bajo que circunstancias se lanzan.
  - Consejos de que hacer cuando se capturen.

- Cuando se decide propagar una excepción, merece la pena dedicar algún tiempo a estudiar si es mejor relanzarla o envolverla. Se recomienda envolver la excepción, si al relanzarla adquiere se puede ampliar su significado.

- A la hora de lanzar una excepción, hay que pensar siempre desde fuera del código que genera las excepciones y no desde dentro, es decir, pensar como se trataría, de que información se dispondrá, de cómo corregir el error.

- Hay que lanzar excepciones con un nivel de abstracción apropiado, expresando el concepto del error y no detalles de implementación.

6. CONCLUSIÓN

Es importante comprender que las excepciones son sólo una herramienta, que no deben utilizarse como ayuda a cumplir con los objetivos descritos en el cuarto apartado.

Los problemas más frecuentes son sin duda:

- Relanzar de forma incontrolada las excepciones de módulos básicos, llegando al punto de no tener prácticamente ninguna información de que originó el error.
- Lanzar excepciones sin medida, llegando al punto de que el control del flujo del código superior, se realiza en base a capturar excepciones lo que dificulta enormemente su mantenimiento y es fuente constante de errores.

7. Fallos cometidos

Hemos localizado dos fallos a la hora de utilizar excepciones en la aplicación, debido a ellos se ha escrito este documento.

1.- Utilización de excepciones para dirigir el código del programa

Por ejemplo, durante el proceso de LogOn se capturan tres excepciones UserNotFound, PasswordError, BloquedUser que son utilizadas para tomar realizar una opción u otra.

Si aplicamos las reglas aquí descritas, un buen diseño del módulo Auth, sólo hubiese lanzado un tipo de excepción AuthException que recubriese adecuadamente errores de conexión JDBC, de implementación... y puesto que por razones de seguridad, normalmente basta con saber si el usuario esta autentificado o no, tan solo habría que devolver True o False y dejar un método disponible para preguntar porque no se ha podido autentificar por si se quisiera información extra.

2.- Excesiva generalización de excepciones.

Algunos módulos de la capa del modelo sólo son capaces de lanzar RemoteException o EJBException, por lo tanto, no se dispone de información suficiente para poder ofrecer un mensaje claro y conciso al usuario.

8. FAQ

¿Qué tipo de excepción debo crear y lanzar?

- Utiliza excepciones comprobadas para casos recuperables y no comprobadas para errores de programación o para informar de precondiciones no cumplidas que imposibilitan continuar con la operación.
- Se suelen utilizar las excepciones no comprobadas (Runtime), ya que facilitan la modularidad y hacen más cómoda la programación, sin embargo, es más fácil perder el control. Por lo que, las comprobadas resultan más adecuadas para programación dirigida a contrato (assert).

¿Cuando lanzo una excepción y cuando retorno un código?

- Los sistemas de excepciones pretende evitar la utilización de valores de retorno complicados. Es recomendable crear excepciones comprobadas ya que se declaran y es obligatorio que se capturen o relancen.
- Si se quiere informar de un error, se suelen utilizar uncheked ya que nadie debe preocuparse de capturarlas o lanzarlas. Sino se confía en que el código que provoque la excepción la capture es recomendable reportarla antes de lanzarla.
- Otra alternativa a los códigos de error complicados, es mantener métodos para consultar el motivo del fallo o que tanto por ciento no se ha completado en caso de no tener éxito.

¿Qué jerarquía debo implementar?

El objetivo de mantener una jerarquía de excepciones, es facilitar la captura de excepciones de distinto tipo.

No hay que complicar en exceso las jerarquías de excepciones, suele ser mejor parametrizar algún constructor.

Un ejemplo de una mala jerarquía es el siguiente:

```
CarException

SportException extends CarException

VerlinaException extends CarException

PickUpException extends CarException
```

Cuando lo correcto sería una excepción cuyo constructor fuese:

```
CarException(String typeCar)
```

¿Donde y cómo se captura una excepción?

Lo antes posible, es decir, lo más cerca posible del punto de origen.

Siempre que se captura una excepción hay que reportarlo de alguna forma, en especial si se va a relanzar.

Bibliografía

Fundamentos JAVA 2: Cay S. Horstmann / Gary Cornell

Internet

Server clinic: Writing good exceptions. Cameron Laird
Best practices in EJB exception handling. Srikanth Shenoy
To check, or not to check? Brian Goetz
Build a better exception-handling framework. Brett McLaughlin

;)

- Si algo puede salir mal, saldrá.
- Si no tenemos éxito, corremos el riesgo de fallar.
- Tenemos derecho a tropezar, pero es nuestra responsabilidad levantarnos de nuevo.
- No existen reglas generales para gestionar excepciones, por eso son excepciones.

APÉNDICE 1

PARTICULARIDADES DE JAVA

Todas aquellas excepciones que extiendan de RuntimeException, son excepciones no comprobadas y el resto se entiende que son comprobadas.

Puesto que en JAVA no se disponen de destructores, a menudo es necesario incluir, el código que libera recursos dentro de bloques finally. Es frecuente que este código pueda lanzar excepciones comprobadas como:

```
connection.close() throws SQLException
```

En este caso, relanzar o envolver la excepción seguramente detendrá el flujo del programa sin necesidad, puesto que las operaciones con la conexión ya se han realizado bien y los posible errores son controlados en otra sección anterior. Capturar la excepción no aporta nada, ya que suele ser imposible corregir el error, que por otro lado no implica ningún fallo del programa.

El API de JAVA puede lanzar algunas excepciones sin sentido como por ejemplo: *EOFException*, una operación que recorre un fichero es normal que en algún momento llegue al final del fichero, es NORMAL, luego no es excepcional. Lo más intuitivo y eficiente para controlar estas operaciones es asegurarnos de que no se ha llegado al final del documento antes de leer y no leer hasta que se lance una excepción.

PARTICULARIDADES DE C++

Se puede decir que todas las excepciones de C++ son no comprobadas, ya que existe un manejador por defecto para las excepciones no declaradas. Esto suele provocar que en aplicaciones de cierta envergadura se pierda el control sobre las excepciones.

El *throw* de C++ aunque similar al de JAVA, se implementa en tiempo de ejecución y no en compilación, por lo tanto, si lanza una excepción no contemplada se invoca a la a función Unexpected y por defecto el programa finaliza informando de la excepción.

C++ solo tiene dos clases fundamentales ***logic_error*** equivalente a las RuntimeException y errores lógicos y las ***runtime_error*** son excepciones impredecibles las equivalentes a Exception y Error.

En C++ se pueden lanzar objetos de cualquier caso, mientras que en JAVA sólo los Throwable, esto provoca que a menudo se utilicen las excepciones para controlar el flujo del programa.

En C++ se puede utilizar "..." para capturar cualquier excepción (catch(...)) esto es porque al contrario que en Java no existe una clase Exception que englobe a todas.

CREAR UNA ACCIÓN

Intranet para un Dpto. Universitario

Cómo crear una acción desde cero

29 de Junio de 2004, v 1.0

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

1. PRÓLOGO

El presente documento trata de dar pinceladas genéricas de cómo crear una acción, para ello trata de simplificar al máximo otros documentos, a los que hace referencia.

Los lectores obtendrán en el presente documento de una forma clara y sencilla los pasos a seguir para crear una acción que le dé más funcionalidad a la aplicación.

Éste documento debe ser leído por todas aquellas personas que vayan a diseñar una acción dentro de la aplicación, los diseñadores del formato de información (creadores de xsl) y desarrolladores en general, ya que aporta la estructura general del funcionamiento de la aplicación.

Las presentes líneas deberán ser leídas después de haberse documentado con los otros textos, ya que requiere un cierto grado de comprensión de temas que aquí no se detallan (no es el cometido que se persigue).

2. INTRODUCCIÓN

En éste documento se trata de que el lector obtenga una visión genérica de cómo realizar una acción y para ello del funcionamiento del modelo-vista-controlador y de struts.

El documento está dividido en varias partes que se detallan a continuación:

Pasos preliminares: acciones a tener en cuenta antes de ponerse en marcha.

Esquema general: primeros pasos que como se puede montar la estructura.

Modelo: qué tener en cuenta a la hora de realizar la parte del modelo.

Controlador: qué tener en cuenta a la hora de realizar la parte del controlador.

Vista: qué tener en cuenta a la hora de realizar la parte del controlador.

¿Cómo funciona la aplicación?: resumen básico de todo.

Conclusión: aspectos importantes.

3. Pasos preliminares

3.1 Análisis

Hay que tener en cuenta que el fin último es que el usuario vea en su navegador la información que busca o que vea opciones que le permitan realizar la funcionalidad deseada de forma rápida y sencilla.

Este aspecto es muy difícil por ello se tiene que responder a una serie de preguntas para poder evaluar, valorar la información que se mostrará en las pantallas, así como el formato que deberán tener éstas. Por ejemplo no es lo mismo hacer una web aislada, donde el formato en principio queda en manos del diseñador, que hacer una parte integrada con una Intranet corporativa donde el formato o el propio diseño de la web viene ya impuesto.

También habrá que preguntarse por aspectos tan importantes como la propia organización de la información, es decir, como se van a mostrar los datos al usuario si en una pantalla o en varias, o una pantalla que contenga varias cosas, etc.

3.2 Diseño

Una vez que tengamos bien estructurada y organizada la información que queremos mostrar, hay que preguntarse, cómo queremos mostrar dicha información, es decir el formato.

Por ejemplo no es lo mismo hacer una web aislada, donde el formato en principio queda en manos del diseñador, que hacer una parte integrada con una Intranet corporativa donde el formato o el propio diseño de la web viene ya impuesto.

Es aquí donde nos preguntamos si queremos por ejemplo tener el logo arriba a la izquierda o si el fondo de la pantalla es verde, el tipo de letra usado, etc.

Aunque el diseño, no es algo primordial en la aplicación en sí (no aporta funcionalidad), es muy importante lo atractiva que le resulte al usuario final, hay que cuidar detalles como la combinación de colores, el tamaño de la letra, tener en cuenta que cuesta menos leer letras minúsculas que mayúsculas, etc. hay todo un mundo sobre éste tema, de hecho hay un montón de profesionales no informáticos que se dedican al estudio y diseño de logos, fuentes, etc.

4. Esquema general

Primero hay que tener en cuenta que se está desarrollando código según el modelo vista-controlador, donde:

Modelo: Parte de la aplicación que se encarga de tratar directamente con el modelo de datos.

Vista: Parte de la aplicación que se encarga del aspecto que se le dan a éstos datos, para que el cliente vea de forma adecuada los mismos.

Controlador: Parte de la aplicación que se encarga de llevar el hilo de la ejecución, es decir, el orden en el cual se mostrarán las páginas.

Visto el modelo de desarrollo, ahora lo que se hace es separar convenientemente cada una de las partes, ubicarlas donde corresponda y configurar la aplicación para que se hagan las llamadas y parseos adecuados.

Para aplicar el modelo de desarrollo descrito se usará Struts (Patrón Java que nos ayudará en las tareas de configuración del hilo de ejecución) y StrutsCX (que es un recubrimiento de Struts que incorpora la gestión de documentación XML).

5. Modelo

Como se ha descrito anteriormente el modelo es la parte del programa que se ocupa de tratar directamente con el modelo de datos, en éste caso se entiende por modelo de datos, la base de datos Oracle que interactúa con la aplicación.

La aplicación no puede tratar directamente con registros, o con campos, la aplicación trata con objetos, para éste fin se crean clases del tipo "ValueObject" que sirven de interfaz entre aplicación y registro, DAO (Data Access Object) o incluso, como es el caso de la aplicación desarrollada Beans, que permiten una eficiencia en el uso de recursos bastante elevada.

También se crearán clases que mediante el uso de las anteriores (ValueObject) permitan automatizar tareas rutinarias como generar listados, realizar consultas, dar altas y bajas, o modificar registros. Éstas clases pueden devolver desde un documento donde se inserta el listado solicitado, o un simple mensaje de éxito o fracaso.

A la hora de crear una acción se tendrá en cuenta el uso de dichas clases a la hora de necesitar (muy probablemente) interactuar con base de datos.

Como se ha comentado antes, se va a usar tecnología XML para la representación de la información, y para el uso de dicha tecnología se va ha hacer uso de StrutsCX. Entonces lo que devuelvan las clases pertenecientes al Modelo serán clases con las cuales StrutsCX pueda generar documentos XML, para ello existen varios métodos:

1. Se puede delegar el hecho de crear el documento XML a partir de una clase a StrutsCX, con la ventaja de la automatización de la tarea y con el inconveniente de no tener el control de cómo se ha generado el documento o de crear un documento no válido.
2. Hacer uso de la clase *Document* (*org.jdom.Document*) que es un estándar y que te permite generar una estructura con clases cuyo paso a XML, es directo.

Si se quiere tener un mayor control del propio XML generado, se recomienda ésta última.

6. Controlador

Como se ha comentado antes el controlador es la parte de la aplicación que se encarga de llevar el hilo de la ejecución, es decir, el orden en el cual se mostrarán las páginas. Para facilitarnos la tarea de llevar el control del flujo de la ejecución se hace uso de Struts.

La forma que tiene Struts de llevar dicho control es mediante el uso de un fichero XML de configuración donde se albergan todas las páginas que componen la aplicación y todo lo que se hace cuando se visita una u otra, el fichero en cuestión se llama struts-config.xml

Un ejemplo de cómo se pude configurar una acción es el siguiente:

```
<!--Muestra la pantalla para insertar un staff-->
<action
        path="/addStaff"
        type="edu.ucm.sip.inet.web.actions.staff.AddStaff">
        <forward name="success" path="/StrutsCXServlet"/>
</action>
```

La primera línea es un comentario que indica que trata la acción posteriormente configurada.

Después se trata ya la acción mediante el uso de la etiqueta <*action*> dentro de la cual se observa que *path* contiene una ruta que se pondrá en el browser o navegador, *type* contiene la clase java a ejecutar en caso de teclear la anterior ruta en el navegador y la última línea lo que dice es que si la acción le devuelve a la *request* (entorno de trabajo de la propia página) "*success*", es decir, éxito que se ejecute StrutsCXServlet que es el encargado de parsear el XML que genera la página con el XSL que tendrá asociado y mostrárselo al navegador (más adelante se detallará éste proceso).

Hay que tener en cuenta que la request no solamente puede devolvernos success, sino también error, failed, etc. y que cada uno de dichos mensajes se puede tratar de forma individualizada.

La clase que se debe ejecutar (en el ejemplo anterior "*edu.ucm.sip.inet.web.actions.staff.AddStaff*") debe heredar de *org.apache.struts.action.Action*, o en nuestro caso de *edu.ucm.sip.inet.web.actions.GeneralAction* que es un recubrimiento de la anterior. Ésta clase obliga a definir un método llamado execute, que será el que se ejecute cuando la aplicación lea en struts-config.xml la ruta y lo redireccione a dicha página.

A continuación se expone una cabecera típica de dicho método:

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
        throws java.io.IOException, javax.servlet.ServletException
```

Dentro de dicho método es conveniente hacer uso de una llamada al método executeAtStart para definir con qué XSL (si se diera el caso) debemos parsear el XML que se generará en la aplicación. Esto está definido en strutsCX-config.xml que se detallará en el siguiente apartado. A continuación se pone un ejemplo de uso de dicho método:

```
executeAtStart("addStaff", mapping, form, request, response);
```

Lo que hace es informar a struts que el XML que hay en la request deberá ser parseado con la entrada llamada “addStaff” que se encuentra en strutscx-config.xml.

Como se ha dicho el párrafo anterior struts lo que hace es parsear el XML de la request con el XSL, para poder hacer esto es necesario insertar “algo” en dicha request. Cuando se dice “algo” se refiere a los métodos citados en la parte del “Modelo” (clase o Document), para insertar se usaría una línea como la siguiente:

```
request.setAttribute("listado", doc)
```

Donde doc es una clase de tipo Document y en donde se guarda la estructura y los datos que compondrán el XML.

Éste doc se supone que nos lo dará alguna clase perteneciente al modelo.

Por último tenemos que decirle a Struts el resultado de la acción para él dictamine el flujo de ejecución, para ello se puede escribir una línea como la siguiente:

```
//Lanzamos la siguiente acción "success" -> ver struts-config.xml
return mapping.findForward(Constants.FORWARD_SUCCESS);
```

Donde Constants.FORWARD_SUCCESS es una constante que contiene “*success”*.

7. Vista

Como se ha comentado antes la vista es la parte de la aplicación que se encarga del aspecto que se le dan a éstos datos, para que el cliente vea de forma adecuada los mismos.

El fin último es la generación de HTML, que es lo que realmente ve el usuario en su navegador, para la generación de dicho HTML, se usa Struts que nos permite parsear en el servidor los datos (XML) y el formato de dichos datos (XSL) generando así el HTML que verá el navegador cliente.

Como se ha visto en el apartado anterior es la propia acción la que dictamina la entrada del fichero strutscx-config.xml que contiene el fichero xsl que se asociará al xml generado.

Un ejemplo de entrada en el fichero strutscx-config.xml sería el siguiente:

<definition name="addStaff">

<put>/web/xsl/addStaff.xsl</put>

</definition>

En el ejemplo se indica que cuando una acción quiera parsear mediante “addStaff” se deberá ir a por “addStaff.xsl”.

En el diseño del XSL se podrá reutilizar las partes que se consideren comunes a toda la aplicación, como la cabecera, el pie de página u otras informaciones como el fondo de pantalla, etc.

Se puede dar el caso de que un xsl tenga botones que digan a la aplicación donde debe ir. A continuación se detalla un ejemplo del caso citado:

<form method="post" action="/coldy/addStaffResul.do"> …

<input type="Submit" value="Insertar Personal" />

Lo que el ejemplo indica es que cuando se pulse en el botón “Insertar Personal” la aplicación cargará en la barra de navegación del browser cliente /coldy/addStaffResul.do, siguiéndose el ciclo de ejecución que se detalla en el siguiente apartado.

8. ¿Cómo funciona la aplicación?

Éste apartado trata de ser un resumen de los anteriores, en el que se harán referencias a temas ya tratados, y se expondrá el flujo de funcionamiento de la aplicación.

Todo empieza cuando el cliente solicita una web a través de su navegador, acto seguido se mira en struts-config.xml qué se debe ejecutar cuando se teclea dicha dirección y se ejecuta el método execute de la clase.

En el método execute de la clase a ejecutar se dice la entrada del fichero strutscx-config.xml que contiene el xsl que dará al formato al documento xml que se va a generar.

También el método execute se hará uso de clases de la parte modelo en caso de tratarse de una acción que necesite de la base de datos, o generará una clase de tipo Document que la insertará en la request.

Por último en la clase execute se meterá en la request el evento (success por defecto) para que sepa struts donde debe continuar o que hacer (para ello volverá a mirar struts-config.xml)

Se puede dar el caso de que el propio xsl contenga un botón y que pulsando sobre dicho botón, sea el propio xsl, el que diga donde se debe ir.

9. Conclusión

Como se puede observar a la hora de realizar una acción hay tres partes bien diferenciadas; modelo, vista, controlador, y por ello es fácil pensar que se pueden desarrollar todas a la vez. Esto es posible siempre y cuando se fijen unos criterios, es decir, se asuman una serie compromisos por cada una de las partes.

Un método de trabajo sería que el diseñador y el desarrollador se pusieran de acuerdo en la estructura XML que une la parte del controlador con la vista.

El diseñador dice lo que necesita para mostrar lo que el usuario quiere ver y junto con el programador de la acción confeccionan un XML, desde el cual el diseñador parte para confeccionar su XSL y al cual debe llegar el programador para que todo funcione.

Otro aspecto importante en la arquitectura MVC (Modelo, Vista, Controlador) es que cada una de las partes no debe interferir en las otras, es decir, el desarrollador de la acción no puede adquirir datos por su cuenta sino que se lo tiene que pedir al modelo, por ello también tiene que haber coordinación entre la parte controlador y modelo para que los datos que uno (controlador) solicita sepa cómo dárselos la otra parte (modelo).

Administración y Desarrollo con

JBOSS 3.2.x

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

Propósito de este documento

Presentación del servidor de aplicaciones web de código abierto JBoss, y dar una serie de guías y ejemplos de configuración básica, suficiente para el uso de dicha herramienta como base para soportar el portal de la Intranet del Departamento de Sistemas Informáticos y Programación. Información adicional puede encontrarse en la documentación de JBoss, disponible por un precio razonable en su web: http://www.jboss.org/

Introducción

JBOSS-Group en uno de los líderes en código abierto basado en la arquitectura J2EE de Java. Una de las principales tecnologías que ofrece JBOSS-Group es JBossServer, que incorpora todas las funcionalidades básicas de un servidor de aplicaciones J2EE. Todas estas funcionalidades son complementadas fácilmente con una serie de extensiones y plug-in's que son fácilmente gestionados mediante una consola de administración que sigue el estándar JMX (Java Management eXtension).

Entre la amplia gama de extensiones disponibles para JBoss encontramos un contenedor de EJB´s compatible con la especificación EJB 2.0, un soporte para mensajería JMS (Java Messaging Service) llamado JBossMQ, JBossTX para transacciones basadas en la JTA (Java Transactional API), JBossCMP para gestionar la persistencia de EJB´s (Container Managed Persistence CMP), JBossSX para proporcionar seguridad basada en el estándar JAAS, y JBossCX para conectividad JCA. El soporte básico para componentes Web como JSP´s y Servlets lo proporciona un motor propiedad de terceros que actualmente puede ser bien Tomcat de Apache, o Jetty.

¿Cómo conseguirlo?

La versión más reciente de JBoss puede conseguirse a través del proyecto SourceForge en la siguiente URL

http://sourceforge.net/projects/jboss

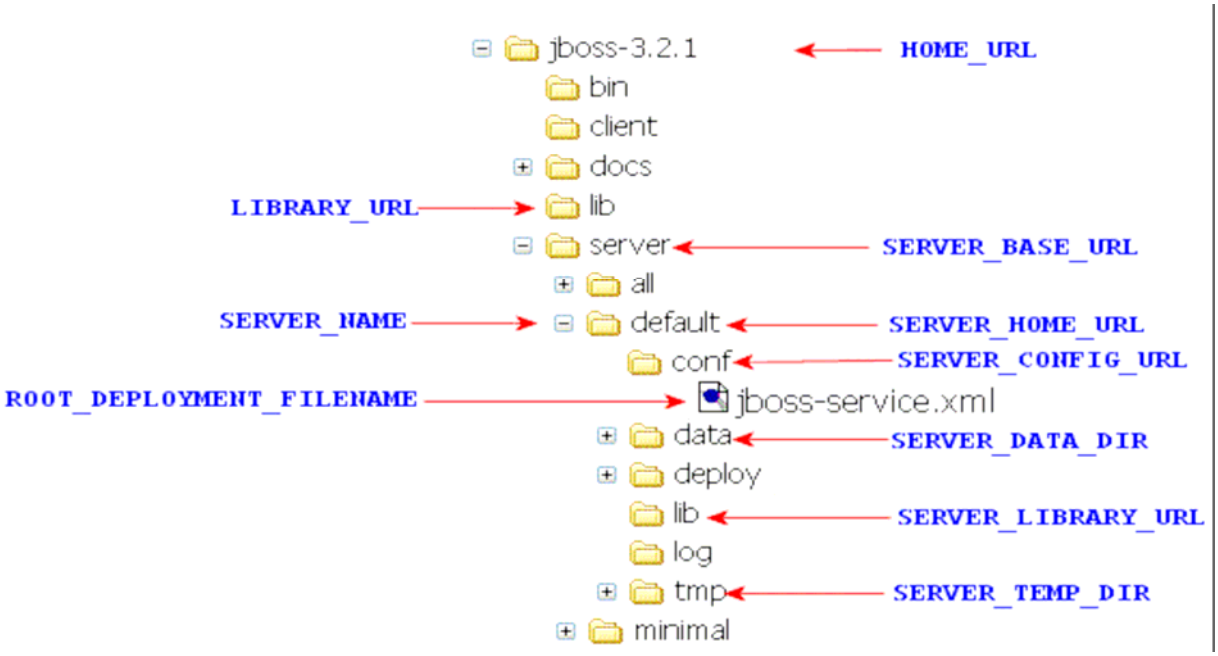
Instalación. Prerrequisitos

Existen distribuciones para diversos sistemas operativos disponibles en la URL anterior. Para que JBoss 3.2.x funcione requiere un JDK 1.3 o superior. Para comprobar que versión del JDK se está usando actualmente se ha de ejecutar *java -version* en la línea de comando. Por ejemplo en un sistema Windows 2000 Server como el utilizado en este proyecto se obtendría la siguiente salida por pantalla:

```
E:\j2sdk1.4.1_02\bin>java -version
java version "1.4.1_02"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.1_02-b06)
Java HotSpot(TM) Client VM (build 1.4.1_02-b06, mixed mode)
```

Una vez obtenido el archivo .jar o .zip de la URL citada anteriormente, se deberá elegir un directorio donde descomprimir JBoss. A diferencia de la mayoría de las aplicaciones basadas en Windows, JBoss no se tiene que instalar. Basta con descomprimirlo en un directorio cualquiera, aunque JBoss-Group recomienda que la ruta absoluta hasta el directorio donde se encuentre JBoss no contenga espacios en blanco, pues puede causar problemas en algunos sistemas operativos.

Una vez descomprimido deberemos encontrar un árbol de directorios prácticamente idéntico al que se muestra a continuación, salvo por la versión de JBoss en el directorio raíz.



En color azul se muestran las variables de entorno que representan los directorios indicados. JBoss usa esas variables internamente, pero también puede resultar muy útil establecerlas como variables de entorno del sistema para realizar tareas de mantenimiento, ya sea mediante el lanzamiento de scripts o mediante el uso de programas de terceros.

Por ejemplo, si se utilizara un script basado en ANT (tecnología que se explicará en otro documento similar a éste) para el lanzamiento de una aplicación, ayudaría mucho establecer las variables JBOSS_HOME y SERVER_HOME porque así el script funcionaría en base a las rutas almacenadas en esas variables y no tendría que ser modificado si se usa en otra máquina, o con otro sistema operativo.

Se va a resumir a continuación la función de los directorios que afectan directamente al desarrollo de este proyecto:

bin : Todas las librerías java básicas para el funcionamiento del servidor se encuentran aquí. También se halla el fichero run.bat que sirve para arrancar JBoss. Para detenerlo, solo se tendrá que ejecutar el fichero batch shutdown con el parámetro –S (que parará servicios adicionales de jboss que se hubieran arrancado) o simplemente cerrar la ventana emergente que se abre al arrancar el servidor pulsando el "aspa" o presionando CTRL+C.

lib : Almacena las librerías que necesita JBoss para arrancar. No se deben poner aquí librerías de terceros o librerías para ninguna de las aplicaciones que funcionen sobre este servidor.

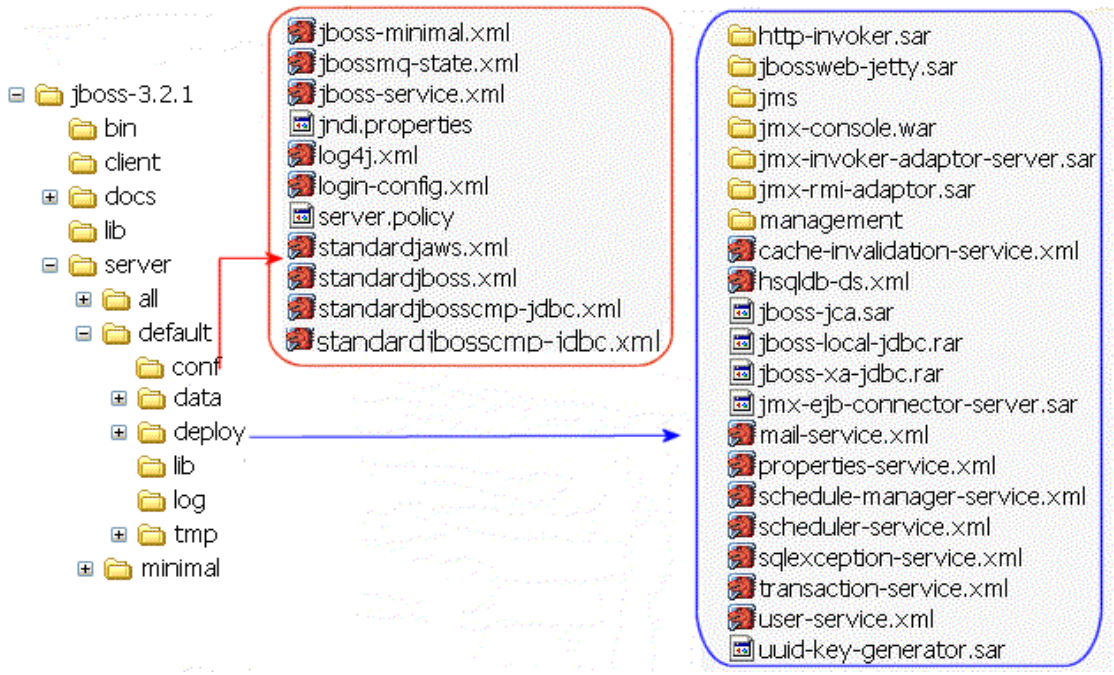
server: Inicialmente este directorio contiene 3 subdirectorios: minimal, default (la configuración por omisión como su propio nombre indica) y all. Estos 3 directorios son configuraciones por defecto que vienen incluidas a modo de ejemplo. Son perfectamente funcionales y sus nombres indican el número de extensiones o funcionalidades adicionales que utilizan. Minimal es una configuración que solo soporta las funciones de servidor de aplicaciones: motor de JSP´s y Servlets, y Logs. No incluye contenedor web, por tanto debería usarse en conjunción con un servidor web independiente como Apache. Default es una configuración que ya incluye contenedor Web, contenedor de EJB´s, gestión de la persistencia, consola de administración JMX vía web...etc. All es la configuración que hace uso de todas las extensiones disponibles.

IMPORTANTE : La elección de la configuración para el funcionamiento de la aplicación es esencial ya que puede darse el caso de que la configuración por defecto (server/default) incluya muchas mas extensiones de las necesarias, con la consiguiente perdida de rendimiento, y mayor consumo de memoria. En el caso de este proyecto se utilizará una versión simplificada del servidor por defecto, a medio camino entre la configuración *minimal y default*.

Dentro de cada directorio de configuraciones, ya sea *minimal, default o all*, se sigue siempre la misma estructura de directorios:

Sea **<config_set>** = *default, all, minimal*, o cualquier otro nombre que se elija para una configuración personalizada de JBoss. Inicialmente solo aparecen estos tres directorios:

<config_set>/conf: contiene el fichero *jboss-service.xml* que especifica los servicios que arranca esta configuración. También incluye ficheros de configuración adicional para dichos servicios Los ficheros más relevantes para el objetivo de este proyecto se explicarán mas adelante según sea conveniente.



IMPORTANTE: Gracias al gran diseño modular de este servidor, con el simple hecho de incluir o no incluir un fichero XML en este directorio se estará activando o desactivando una funcionalidad o extensión de JBoss. Así para pasar de una configuración *all* a una configuración *minimal* solo habría que eliminar de este directorio, o añadir en el caso contrario, una serie de ficheros XML elegidos cuidadosamente.

<config_set>/deploy: El directorio de despliegue, donde se depositarán los ficheros WAR o ear de las aplicaciones que quieran hacer uso de JBoss.

<config_set>/lib: Librerías utilizadas por la aplicación que esté funcionando en esta configuración. Aquí se incluirían por ejemplo los drivers de los diversos fabricantes de SGBD que se vayan a utilizar en la aplicación. (Por ejemplo classes12.zip de Oracle-JDBC)

Una vez se haya ejecutado una aplicación en esta configuración del servidor aparecerán los siguientes directorios:

<config_set>/work, <config_set>/data y <config_set>/tmp: Con los ficheros temporales que son utilizados durante el funcionamiento de la aplicación: Servlet's, JSP´s, EJB´s y otros componentes en ejecución,

<config_set>/log: Los logs de arranque y funcionamiento de esta configuración.

El Contenedor Web: Tomcat

JBoss se distribuye actualmente con Tomcat 4.1.x como contenedor web por defecto. El servicio integrado de Tomcat es el directorio *jbossweb-tomcat41.sar* que encontraremos en el directorio deploy de cada una de las diferentes configuraciones (<config_set>/deploy). Todos los ficheros que Tomcat necesita para funcionar se encuentran en ese directorio, como también el fichero web.xml que proporciona una configuración por defecto para las aplicaciones web.

Para configurar Tomcat con JBoss se deberá editar el fichero *META-INF/jboss-service.xml* dentro del directorio <config_set>/deploy/*jbossweb-tomcat41.sar* .

Dentro de la etiqueta **<attribute name="Config"> .... </attribute>** se encuentran los datos de configuración básicos para el servidor Web. Incluye la información sobre el conector HTTP (en el puerto 8080 por defecto), sobre el conector AJP en el Puerto 8009 (que se usa si se quiere conectar a través de un servidor web como Apache) y un ejemplo de cómo configurar una conexión de puertos seguros mediante SSL (Secure Socket Layer). Es aquí donde se cambiará el puerto HTTP por el puerto 80 (puerto HTTP por defecto) de modo que sea el servidor que escuche las conexiones dirigidas al puerto por defecto de la máquina.

http://maquina:8080/ → http://maquina/ (puerto 80 implícito)

Los Logs de Tomcat pueden ser encontrados en el directorio *<config_set>/log*.

Construyendo una configuración a Medida

Dado que con el simple hecho de eliminar o añadir ciertos ficheros xml en el directorio conf de cada <config_set> podemos activar y desactivar extensiones del servidor, podemos aprovechar para optimizar el rendimiento de JBoss a la medida de nuestras aplicaciones. Se puede partir de la configuración default y crear una configuración adaptada a las necesidades de una determinada aplicación en cuestión de minutos.

Primero se copia el contenido del directorio default a otro directorio dentro de JBOSS_HOME/server. En principio solo son necesarios los directorios default/lib, default/deploy y default/conf. El resto no hace falta que se copien pues se generan automáticamente al arrancar el servidor.

Una vez realizado este paso se debería tener una estructura de directorios semejante a esta: JBOSS_HOME/server/miConfig/deploy
/lib
/conf

Supongamos que la aplicación que vamos a probar esta encapsulada en un fichero EAR en el directorio deploy, y que hemos añadido las librerías de Oracle y cualquier otra librería necesaria en tiempo de ejecución en el directorio lib. Ahora, sin

haber hecho ninguna modificación todavía, se debería arrancar esta nueva configuración mediante el comando %JBOSS_HOME%/bin/run.bat -c miConfig para comprobar que la copia funciona correctamente

Una vez comprobado que la aplicación funciona correctamente en la nueva configuración (que ahora debe ser idéntica a la configuración default), ya se puede comenzar a modificar el número de extensiones de las que hace uso esta configuración.

Servicios Fundamentales

Estos servicios se configuran a través del fichero **<config_set>/conf/*jboss-service.xml***. Adicionalmente pueden ir acompañados de una serie de ficheros de configuración específicos para esos servicios. Algunos de estos ficheros son:

Jboss-minimal.xml: Configuración de los servicios mínimos necesarios en todo servidor JBoss.

Standard-jboss.xml: Configuración por defecto de los servicios para EJB´s.

Web.xml: Aquí se definen valores que van a ser compartidos por todas las aplicaciones que funcionen bajo esta instancia de Tomcat (Que hace las veces de contenedor Web para JBoss). Este será el primer fichero Web.xml en ser analizado, seguido del WEB-INF/web.xml de cada aplicación. Aquí configuraremos el timeout de sesión del servidor web.

Jndi.properties: Configuración del Protocolo JNDI (Java Naming and Directory Interface) para acceso a los servicios de directorio que sirven para localizar cualquier componente J2EE mediante su nombre JNDI en entornos distribuidos. No se recomienda editarlo ni modificarlo.

Log4j.xml : Fichero utilizado para configurar el sistema de Logs de JBoss. Log4j es el sistema de Logs basado en Java mas completo y más utilizado en la actualidad. Es altamente configurable y por ello requiere un fichero xml completo para esta funcionalidad.

Login-config.xml : Este fichero define los dominios de seguridad. Se usa en aplicaciones que implementan el estándar JAAS (Java Authentication and Authorization)de seguridad. En este caso, no es necesario para el proyecto.

Servicios Adicionales

Los servicios no fundamentales, que pueden ser desplegados y modificados "en caliente" se encuentran definidos en cada directorio **<config_set>/deploy** mediante descriptores XML o ficheros SAR (Jboss Service Archive), que incluyen los descriptores y otros recursos y librerías adicionales necesarios para el funcionamiento del servicio. Los ficheros y directorios de mayor interés para este proyecto son:

<config_set>/deploy/*jbossweb-tomcat41.sar* : Directorio que representa el fichero .SAR de configuración de Tomcat, desplegado para su modificación en caliente. Aquí se configurarán aspectos fundamentales de Tomcat como los conectores, y el puerto de escucha HTTP.

Oracle-ds.xml : Aunque el sufijo -ds recuerde a DataSource (Clase Java que representa una fuente de datos) y puedan confundirse con ficheros que únicamente sirvan para configurar el acceso a BBDD, todos los ficheros acabados en -ds representan configuraciones de servicios de conectividad. En este caso el fichero oracle-ds ha de ser creado, ya que por defecto JBoss utiliza su propio SGBD, Hypersonic. Dentro de este fichero se define el tipo de driver, usuario, password, numero máximo de conexiones simultáneas, tamaño del "pool" de conexiones...etc.

He aquí un ejemplo de Oracle-DS.xml básico, que configura un DataSource para Oracle con definición de pool de conexiones y timeout de conexión.

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
 <local-tx-datasource>
  <jndi-name>OracleDS</jndi-name>
  <connection-url>jdbc:oracle:thin:@maquina:puerto:nombre_bd</connection-url>
  <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
  <user-name>usuario</user-name>
  <password>password</password>
  <min-pool-size>10</min-pool-size>
  <max-pool-size>40</max-pool-size>
  <idle-timeout.minutes>5</idle-timeout.minutes>
 </local-tx-datasource>
</datasources>
```

Una vez configurado este descriptor, solo harían falta los drivers JDBC en el directorio <config_set>/lib para que las conexiones a la Base de Datos seleccionada funciones. En caso de utilizar otro tipo de SGBD se deberá cambiar el Oracle-DS.xml por el fichero correspondiente al SGBD que se vaya a utilizar, y colocar los drivers de dicho SGBD en el directorio <config_set>/lib.

En resumen, una configuración básica como la que ha sido utilizada en este proyecto estará compuesta por:

- Los ficheros: **"jboss-minimal.xml", "jboss-service.xml", "jndi.properties", "log4j.xml", "standardjboss.xml" y "web.xml"** en el directorio **<config_set>/conf.**

- Los ficheros: **"jboss-jca.sar", "jboss-local-jdbc.rar", "jboss-xa-jdbc.rar", "oracle-ds.xml", "transaction-service.xml"** en el directorio **<config_set>/deploy**

- El directorio **<config_set>/deploy/*jbossweb-tomcat41.sar***

Con lo que se obtienen los servicios de Contenedor Web (HTML, Servlets, JSP's), Contenedor de EJB´s, servicios transaccionales para conexión a Base de Datos, servicio de nombres y directorios mediante JNDI y sistema de Logs mediante Log4j. Por lo tanto se han descartado un gran número de funcionalidades de JBoss que no son necesarias en este proyecto y que ralentizarían el funcionamiento de la aplicación innecesariamente

Preguntas más frecuentes

¿Cómo arrancar JBoss como servicio en Windows?

El método más común y más conocido para arrancar JBoss como un servicio es mediante el uso de otra herramienta: Alexandria's JavaService. JavaService es una utilidad Java de libre distribución que arranca cualquier archivo JAR como servicio.

Se puede descargar gratuitamente en la URL siguiente: http://www.alexandriasc.com/software/JavaService/index.html

- Establecer como variables de entorno del sistema JBOSS_HOME y JAVA_HOME apuntando a los correspondientes directorios de instalación del JDK y de JBoss.

- Copiar JavaService.exe en el directorio JBOSS_HOME/bin

- Instalar el servicio Windows ejecutando los siguientes comandos:

%JBOSS_HOME%\bin\JavaService.exe -install JBossServerService %JAVA_HOME%\jre\bin\server\jvm.dll -Xmx256m -Xms256m -Xincgc -Djava.class.path=%JAVA_HOME%\lib\tools.jar;%JBOSS_HOME%\bin\run.jar -start org.jboss.Main -stop org.jboss.Main -method systemExit -out %JBOSS_HOME%\server\default\log\stdout.log -err %JBOSS_HOME%\server\default\log\stderr.log -current %JBOSS_HOME%\bin

- Si se desea eliminar el servicio ejecutar el siguiente comando:

%JBOSS_HOME%\bin\JavaService.exe -uninstall JBossServerService

- Después acceder a los servicios de la máquina con S.O. Windows, seleccionar el servicio llamado JBossServerService y configurar su método de arranque como AUTOMÁTICO, arrancar el servicio y aplicar los cambios. La próxima vez que se arranque la máquina JBoss arrancará como un servicio más.

NOTA: Al arrancar JBoss como servicio, no se abre ninguna ventana de consola con información de trazas por pantalla. Si se desean consultar los logs del servidor habrá que acudir al directorio <config_set>/log.

¿Cómo se indica cuál es la aplicación por defecto dentro de una misma configuración?

Dentro de cualquier directorio <config_set>/deploy del servidor, puede haber numerosas aplicaciones diferentes funcionando simultáneamente. Cada aplicación tiene su propio contexto web, es decir, una forma de identificarla en la URL, por ejemplo:

http://maquina:8080/app1/index.html → app1 es el contexto web de una aplicación llamada Aplicacion1 y desplegada en un fichero aplicacion1.ear en el directorio deploy.

Simultáneamente puede haber otra aplicación en el directorio deploy con el nombre Aplicación2 y que esta encapsulada en el fichero app2.war. Su contexto web ha sido indicado en un fichero de configuración y es contabilidad de modo que aunque su fichero encapsulado se llama aplicacion1.ear, la URL completa que habrá que introducir para acceder a esta aplicación será:

http:://maquina/contabilidad

El contexto web de una aplicación se toma del nombre del fichero WAR o EAR solo si no se indica otro nombre distinto en alguno de los siguientes descriptores de despliegue de la aplicación La aplicación por defecto es aquella que no hace uso del contexto web, es decir, que se la invoca automáticamente cuando no se indica un contexto web:

http://maquina:8080/ o bien
http://maquina/ → si el conector http escucha en el puerto 80

Hay tres formas para configurar una aplicación por defecto en JBoss:

- *La manera Estándar*: En el fichero **METAINF/application.xml** dentro del fichero.ear, se pone <context-root>/</context-root>.

- *La manera JBoss* : Añadir el fichero jboss-web.xml en el directorio WEB-INF del fichero.war de la aplicación y especificar allí el <context-root>/</context-root>.

- *La manera de Tomcat* : Llamar al fichero ROOT.war ó ROOT.ear

IMPORTANTE : La configuración incluida en el fichero META-INF/application.xml tendrá precedencia sobre cualquier otra incluida en el WEB-INF/jbossweb.xml. Llamando al fichero ROOT.war tendremos precedencia sobre cualquier otra configuración incluida en descriptores de despliegue.

Desarrollo con ANT

Intranet SIP

Ventajas que aporta el uso de ANT en el proyecto

04 de Junio de 2004, v1.1

Daniel Fonseca
Gustavo Romero
Mariano Herrera

ÍNDICE

1. PRÓLOGO

Este documento esta recomendado para todos aquellos que vayan a iniciar un proyecto basado en JAVA, como para aquellos que deban mantenerlo.

Para comprender el documento solo se requieren conocimientos básicos de JAVA.

2. INTRODUCCIÓN

Este documento aborda tres cuestiones bien diferenciadas:

- Presentación de ANT como herramienta de gran utilidad tanto durante el desarrollo, puesta en marcha y mantenimiento de un proyecto. Apartado 3.
- Guía básica para empezar a utilizar la herramienta. Referencias sobre el funcionamiento, ejemplos de configuración básica. Apartado 4.
- Uso de ANT durante el desarrollo de la Intranet del Departamento de Sistemas Informáticos y Programación. Como ha facilitado ANT el desarrollo del proyecto y que puede ayudar a su puesta en marcha y posterior mantenimiento. Apartado 5.

Lo que no se pretende es ofrecer una guía de referencia sobre la herramienta de Apache, ni enumerar sus ventajas. Información adicional puede encontrarse en la documentación oficial **[ANT01]**, disponible gratuitamente.

3. Presentación de ANT

3.1 ¿Qué es ANT?

Apache Ant es una herramienta desarrollada en Java similar a otro tipo de herramientas tipo "make", para la compilación y construcción de proyectos.

A pesar de la gran cantidad de utilidades de tipo Make que hay en el mercado (incluso de libre distribución), Ant sigue destacando sobre las demás porque es multiplataforma, mientras los otros programas Make son inherentemente basados en recubrimientos o entornos de ejecución de ciertos sistemas operativos (Shells). Con esas otras utilidades, todo proyecto necesitaría de varios archivos Make para su compilación y construcción en otros tantos Sistemas Operativos en los que se quisiese probar un proyecto, con la consiguiente pérdida de tiempo y de esfuerzo. No obstante otras herramientas como Maven **[MAV01]** se están imponiendo en algunos sectores.

Otra ventaja adicional es la sintaxis. Todas las herramientas tipo Make tienen una sintaxis especial, y casi siempre muy estricta. Esa rigidez ocasionaba innumerables errores en la ejecución de dichos problemas e incluso aparecían nuevos errores dependiendo del sistema Operativo donde se ejecutasen. Ant utiliza ficheros XML para indicar que es lo que quiere hacer, como se quiere hacer y en que orden o con qué condiciones se ha de hacer. Por tanto utiliza una sintaxis que va camino de ser cuasi-universal en la informática de este siglo.

3.2 ¿Cómo conseguirlo?

Hasta la fecha la última versión ANT 1.6.1 está disponible en la Web de Apache en la siguiente URL: http://ant.apache.org/bindownload.cgi

3.3 Instalación

Ant funciona correctamente en Linux, Unix (Solaris y HP-UX), Windows 9x y NT, OS/2, Novell Netware 6 y MacOS X.

Se necesita un parser de XML que cumpla el estándar JAXP **[JAXPSUN]** instalado y disponible en el Classpath, tanto para instalar Ant, como compilar el código fuente.

La distribución binaria incluye el Parser de XML Xerces 2 de Apache. Si se desea usar otro Parser de XML, debe eliminar los ficheros `xercesImpl.jar` y `xml-apis.jar` del directorio /lib de Ant y poner en ese directorio, o en cualquier otro directorio dentro del Classpath, las librerías .jar del nuevo Parser.

Para la versión más reciente de Ant se necesita además el JDK 1.2 o superior de Java instalado.

Nota Importante 1: La máquina virtual de Java de Microsoft no esta soportada.

Nota Importante 2: Si no hay JDK instalado, sino solo el entorno de ejecución de Java (JRE) algunas tareas de Ant explicadas en este documento no funcionan.

Antes de ejecutar Ant quedan unos últimos detalles que deben completarse:

- Añadir el directorio ANT_HOME/bin al Path del Sistema o del usuario.
- Crear la variable de entorno `ANT_HOME` y asignarle el valor del directorio donde se haya instalado Ant.

Cuando ANT ejecuta sus tareas interpreta algunas variables de entorno por eso es recomendable tener la variable `JAVA_HOME` apuntando al directorio de instalación de la JDK.

4. Guía Básica de ANT

4.1 Estructura Básica de un Proyecto Ant

En el directorio ANT_HOME/bin encontramos el ejecutable ant.bat. Este ejecutable busca un fichero de construcción en el directorio de trabajo, por defecto *build.xml* e intenta leerlo como archivo de tareas a realizar.

4.2 Escribiendo un Archivo de Construcción Sencillo

Los archivos de Construcción de Ant (en adelante se utilizará el término buildfiles) están escritos en XML. Cada buildfile contiene un proyecto Ant y al menos un Objetivo (en adelante Target). Los Targets contienen elementos Tarea (Task). Cada elemento Tarea de tiene un atributo id único para que pueda ser referenciado de forma unívoca.

Proyectos

Un proyecto Ant tiene 3 atributos:

Atributo	Descripción	Requerido
name	El nombre del proyecto.	No
default	El *Target* por defecto en caso de que no se indique un *target* al invocar al proyecto	Si
basedir	El directorio base desde el que se calculan todos las rutas relativas. Este atributo puede ser sobrescrito fijando la propiedad basedir de antemano. Cuando se haga esto , se debe omitir el atributo de la etiqueta Project Si no se indican ni la propiedad ni el atributo se toma como basedir el directorio en el que resida el buildfile.	No

Targets

Un *Target* puede depender de otros *Target*. Esto se indica mediante el atributo *depends* de la etiqueta *Target*. En este atributo se indican los id de las tareas que es necesario ejecutar antes de realizar la tarea en cuestión, y en el orden (de izquierda a derecha) en el que deben ser ejecutados para que todo funcione correctamente.

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C,B,A"/>
```

Supongamos que se quiere ejecutar la tarea D. Si se mira su atributo *depends* veremos que depende de las tareas C, B y A en ese orden. Pero si se observa cuidadosamente la lista de dependencias de C, B y A veremos que en realidad las tareas no se ejecutan en ese orden. C depende de B, y B depende de A, así que primero se ejecuta A, después B y por último C, entonces D puede ejecutarse correctamente. Un *Target* se ejecuta una única vez incluso si varios *Targets* dependen de ella.

Por ejemplo se puede tener un *Target* para compilar la aplicación, y otro para construir un fichero WAR o EAR. No se pueden crear dichos ficheros si no se ha compilado primero,

luego tiene sentido hacer que la tarea para construir el fichero WAR o EAR dependa de que la tarea de compilación se haya realizado antes. Ant se encarga de resolver estas dependencias. Si Ant detecta que una tarea depende de otras tareas, pero que esas otras tareas no es necesario ejecutarlas porque ya han sido ejecutadas con anterioridad, Ant no ejecutará dichas tareas, lo que no afectará al resultado de la tarea principal.

Un *Target* puede ser ejecutado de manera condicional, es decir, si cierta propiedad o propiedades han sido fijadas entonces esta tarea puede ejecutarse. Esto se realiza mediante los atributos *if* y *unless*. Si ninguno de estos atributos esta presente, la tarea se ejecutará incondicionalmente.

Un *Target* tiene los siguientes atributos:

Atributo	Descripción	Requerido
name	El nombre del *Target*.	Si
depends	Una lista de valores separados por coma de los *targets* de los que depende	No
if	El nombre de la Propiedad que debe estar fijada para poder ejecutar esta tarea	No
unless	El nombre de la propiedad que debe No estar fijada para poder ejecutar esta tarea	No
description	Una corta descripción de la tarea. Esta descripción saldrá por pantalla si usamos la opción de ejecución -verbose	No

Es posible nombrar a algunos *targets* empezando por guión como por ejemplo `"-reinicia"` , y pueden ser usados para nombrar *targets* que NO deben ser llamados desde la línea de comandos.

Tasks

Una Tarea (*Task*) es un bloque de código que se quiere ejecutar.

Una tarea puede tener múltiples atributos, y pueden ser considerados como argumentos de una función. El valor de un atributo puede contener referencias a una Propiedad. Estas referencias serán resueltas antes de que la tarea sea ejecutada. Una estructura habitual de una tarea es de la forma :

```
<nombre atributo1="valor1" atributo2="valor2" ... />
```

Existe un conjunto de tareas predefinidas y otro gran conjunto de tareas opcionales, además de todas las tareas que pueden ser escritas por el usuario. Estas tareas se cubren ampliamente más adelante en este documento.

Properties

Un proyecto puede contener un conjunto de propiedades (*Properties*) que pueden ser fijadas en el *buildfile* mediante la etiqueta <*property*> o pueden fijarse fuera de Ant.

Cada *property* tiene nombre y valor y el nombre es sensible a mayúsculas. Pueden usarse en los valores de los atributos de las Tareas. Esto se consigue escribiendo el nombre de la propiedad entre `"${"` y `"}"` en el valor del atributo. Por ejemplo, si existe una propiedad "builddir" con el valor "build", entonces podría ser utilizado en un atributo de

esta manera: `${builddir}/classes`. Este valor se resuelve en tiempo de ejecución como "`build/classes`".

Properties Predefinidas

Ant proporciona acceso a todas las propiedades del sistema como si hubiesen sido definidas usando la Tarea *Property*. Por ejemplo, `${os.name}` se resuelve como el nombre del Sistema Operativo.

Para obtener una lista de todas las propiedades del sistema consúltese la documentación del método getProperties() de la clase System de Java. (System.getProperties())

Además de las propiedades de Sistema, Ant tiene algunas propiedades predefinidas mas:

basedir: La ruta absoluta del directorio base del proyecto (como haya sido definida en el atributo basedir de la etiqueta <project>).

ant.file: El path absoluto del buildfile.

ant.version: La versión de Ant

ant.project.name: El nombre del proyecto que se esta ejecutando; Se le asigna el valor en la etiqueta <project>.

ant.java.versión: La Java Virtual Machine que Ant ha detectado; Actualmente reconoce las versiones "1.1", "1.2","1.3" and "1.4".

Tareas Predefinidas

Mkdir : Crea el directorio especificado en el atributo dir.
Sintaxis : <mkdir dir="nombreDir"/>

Fileset : Representa un conjunto de ficheros. Se usa habitualmente para realizar una acción sobre un conjunto de ficheros. Mediante el atributo dir se indica el directorio en el que se encuentran los ficheros o directorios que se quiere representar. Mediante la etiqueta anidada <include> se define que ficheros del directorio indicado por el atributo dir se han de contemplar. El doble asterisco representa cualquier cantidad de subdirectorios anidados, mientras que un único asterisco representa cualquier cantidad de caracteres en el nombre de un fichero o directorio.

```
Sintaxis : <fileset dir="${lib}">
                    <include name="**/*.jar"/>
           </fileset>
```

Pathelement : Representa una ruta, un directorio, y todos sus archivos
Sintaxis : <pathelement location="${tlds}"/>

Copy : Copia un único fichero o un conjunto de ficheros de un origen a un destino.

Sintaxis:

- Varios Ficheros :

```
<copy todir="${jarDir}">
  <fileset dir="${classdir}" includes="**/*.class" />
   <fileset dir="${classdir}" includes="**/*.properties"/>
</copy>
```

- Un Fichero :

```
<copy file="${build.dir}/${earFile}"
        todir="${jboss.home}/server/default/deploy"/>
```

Delete : Borra un fichero o directorio indicado por sus atributos file o dir

Sintaxis :

- Fichero : <delete file="${jboss.home}/server/default/deploy/coldy.ear"/>
- Directorio : <delete dir="${build.dir}/deploy"/>

Javac : Compila una serie de ficheros Java. Sus atributos indican el directorio fuente y destino, qué ficheros se quiere compilar. Debe indicarse el classpath utilizado para la compilación. Si se quieren excluir ficheros de la compilación se deben indicar expresamente mediante la etiqueta <exclude>. Para mas detalle en los atributos de la Tarea Javac consultar el manual de Ant.

```
Sintaxis:
  <javac srcdir="${src}" destdir="${build.classes.dir}"
                debug="on"
                deprecation="on"
                optimize="off"
                includes="**/*.java">
       <classpath refid="classpath"/>
      <exclude name="com/"></exclude>
      <exclude name="org/"></exclude>
  </javac>
```

jar : Crea un fichero Jar con las clases Java de un directorio indicado

```
Sintaxis :
  <jar jarfile="${build.dir}/${jar.filename}">
        <fileset dir="${build.classes.dir}" includes="**/*.class"/>
        <metainf dir="${deploymentdescription}/jar/"
              includes="jboss.xml,ejb-jar.xml"/>
```

war : Crea un fichero War con los recursos necesarios y descriptor de despliegue

```
Sintaxis :
  <war warfile="${build.dir}/${war.filename}"
   webxml="${deploymentdescription}/web/web.xml">
```

ear : Crea un fichero Ear con el fichero jar, war y el descriptor de despliegue.

```
Sintaxis:
    <ear earfile="${buil.dir}/${ear.filename}"
    appxml="${deploymentdescription}/ear/application.xml"
    basedir="${build.dir}"
    includes="${jar.filename},${war.filename}"/>
```

sql : Lanza un script SQL, utilizando JDBC

```
Sintaxis :
    <sql driver="${jdbc.driver}" url="${jdbc.url}"
            userid="${jdbc.user}" password="${jdbc.password}"
            delimiter=";" print="true" onerror="continue">
            <classpath>
                    <pathelement location="${jdbc.driver.jar}"/>
            </classpath>
            <fileset dir="${sql.dir}">
                    <include name="${insert_intdata.id}.sql"/>
            </fileset>
    </sql>
```

5. Aplicación de ANT

A lo largo del desarrollo del proyecto se ha utilizado ANT para facilitar el despliegue y prueba del código, así como para instalar las distintas versiones en el servidor.

Durante el desarrollo se distinguen dos actividades fundamentales:

- Desarrollo de código: Mientras se desarrolla el código es necesario, codificar, compilar y probar la aplicación. Debido a las características del proyecto, este proceso no es sencillo, ni rápido por lo que es necesario automatizarlo.
- Instalación de versiones: Una vez se tiene una versión estable, es necesario instalarla, lo que conlleva desinstalar la versión antigua, crear el modelo de datos, rellenarlo... Si se tiene, en cuenta que este proceso será necesario repetirlo en varias máquinas (entornos de pruebas, de producción...) es conveniente automatizarlo, además así se facilita la migración a otro servidor.

Para cada una de estas dos actividades se han utilizado dos buildfiles, compuestos de las tareas necesarias. A continuación se explican brevemente dichas actividades, para mayor detalle dirigirse a los comentarios de los buildfiles.

5.1 Desarrollo

Las tareas necesarias para automatizar el proceso de desarrollo son las siguientes:

- Compilar
- Generar el fichero de despliegue EAR
- Desplegar la aplicación.

Cada una de estas tareas está representada por una task del fichero *buid.xml*, para lanzarlas todas basta con ejecutar la tarea ***all***.

Para poder ejecutar las distintas tareas son necesarias las siguiente propiedades, que se inicializan en la task "***init***":

java.home: el valor por defecto lo toma de la variable de entorno *JAVA_HOME*.

j2ee.home: el valor por defecto lo toma de la variable de entorno *J2EE_HOME*.

jboss.home: el valor por defecto lo toma de la variable de entorno *JBOSS_HOME*.

dirs.base: Directorio de origen lo toma del argumento externo *basedir*.

Otros directorios como los Fuentes (*src*)… son inicializados a partir de este según la estructura estándar de una aplicación web.

build.dir: Directorio de destino, por defecto es el directorio basedir}/build.

Es un directorio temporal para guardar los ficheros de despliegue generado:

jar.filename valor por defecto "coldy.jar"

war.filename valor por defecto "coldy.war"

ear.filename valor por defecto "coldy.ear"

Antes de copiarlos al servidor al directorio especificado por ***deploy.dir*** cuyo valor por defecto es "*JBOSS_HOME/server/default/deploy*".

5.2 Instalación

Las tareas necesarias para automatizar el proceso de desarrollo son las siguientes:

- Crear el modelo de datos.
- Destruir el modelo de datos.
- Añadir los datos por defecto a las tablas.
- Borrar los datos de las tablas.

Cada una de estas tareas está representada por una task del fichero *setup.xml*, para lanzarlas todas basta con ejecutar la tarea ***all***.

Tanto la creación y destrucción del modelo y la inserción y eliminación de datos, están controlados por pequeños scripts sql, lo que facilita su gestión y manejo. Para lanzar todos estos scripts juntos, se utiliza una utilidad implementada en JAVA que fusiona los scripts que se le ordene.

Bibliografía

[JAKA01]
http://jakarta.apache.org/

[ANT01]
http://ant.apache.org/manual/

[MAV01]
http://maven.apache.org/

[JAXPSUN]
http://java.sun.com/xml/

APÉNDICE 1

1. Introducción

A continuación se exponen los dos ficheros de configuración utilizados durante el desarrollo del proyecto.

2. build.xml. Compilación y Despliegue

```
<?xml version="1.0"?>

<project name="Jboss 3.2.3 COLDY" default="all" basedir=".">


    <!-- Inicializa las variables que se utilizan en el script
         a partir de las variables de entorno del S.O. -->
    <target name="init">

        <!-- Varibles de Entorno -->
        <property environment="env"/>

        <property name="java.home" value="${env.JAVA_HOME}"/>
        <property name="j2ee.home" value="${env.J2EE_HOME}"/>
        <property name="jboss.home" value="${env.JBOSS_HOME}"/>


        <!-- Directorios de Origen -->
        <property name="dirs.base" value="${basedir}"/>

        <property name="src" value="${dirs.base}/src"/>
        <property name="lib" value="${dirs.base}/WEB-INF/lib"/>
        <property name="tlds" value="${dirs.base}/WEB-INF/tlds"/>
        <property name="classbase" value="${dirs.base}/WEB-INF/classes"/>

        <property name="web" value="${dirs.base}/web"/>
        <property name="xsl" value="${dirs.base}/web/xsl"/>
        <property name="xml" value="${dirs.base}/WEB-INF/xml"/>
        <property name="resources" value="${dirs.base}/resources"/>

        <property name="deploymentdescription"
value="${dirs.base}/deploymentdescriptors"/>

        <!-- Directorios de Destino -->
        <property name="build.dir" value="${basedir}/build"/>
        <property name="build.classes.dir" value="${build.dir}/classes"/>

        <property name="jar.filename" value="coldy.jar"/>
        <property name="war.filename" value="coldy.war"/>
        <property name="ear.filename" value="coldy.ear"/>

        <property name="deploy.dir" value="${jboss.home}/server/default/deploy"/>
    </target>

    <!-- Creamos los Directorio Destino -->
    <target name="create" depends="init">
        <mkdir dir="${build.dir}"/>
```

```
        <mkdir dir="${build.classes.dir}"/>
    </target>

    <!-- Compilamos los ficheros fuentes Java -->
    <target name="compile" depends="create">

        <!-- Build classpath -->
        <path id="classpath">

            <fileset dir="${lib}">
                <include name="**/*.jar"/>
            </fileset>

            <fileset dir="${jboss.home}/client">
                <include name="**/*.jar"/>
            </fileset>

            <fileset dir="${java.home}/lib">
                <include name="**/*.jar"/>
            </fileset>

            <fileset dir="${j2ee.home}/lib">
                <include name="**/*.jar"/>
            </fileset>

            <pathelement location="${tlds}"/>
            <pathelement location="${resources}"/>
            <pathelement location="${build.classes.dir}"/>

            <!-- So that we can get jndi.properties for InitialContext -->
            <pathelement location="${basedir}/jndi"/>
        </path>

        <property name="build.classpath" refid="classpath"/>

        <javac srcdir="${src}"
            destdir="${build.classes.dir}"
            debug="on"
            deprecation="on"
            optimize="off"
            includes="**/*.java">
            <classpath refid="classpath"/>
            <exclude name="org/"></exclude>
            <exclude name="com/"></exclude>
        </javac>

        <copy todir="${build.classes.dir}">
            <fileset dir="${src}" includes="**/*.properties"/>
        </copy>

    </target>

    <target name="generateJAR" depends="compile">
        <jar jarfile="${build.dir}/${jar.filename}">
            <fileset dir="${build.classes.dir}" includes="**/*.class"/>
            <fileset dir="${build.classes.dir}" includes="**/*.properties"/>
        <!--fileset dir="${build.classes.dir}">
              <include name="${appname}.class" />
              <include name="${appname}Home.class" />
              <include name="${appname}EJB.class" />
        </fileset-->
        <metainf dir="${deploymentdescription}/jar/" includes="jboss.xml,ejb-
jar.xml"/>
```

```
      </jar>
    </target>

    <target name="generateWAR" depends="compile">
       <war warfile="${build.dir}/${war.filename}"
webxml="${deploymentdescription}/web/web.xml">
           <webinf dir="${deploymentdescription}/web/">
               <include name="*.xml"/>
           </webinf>
         <classes dir="${build.classes.dir}">
             <include name="**/*.class"/>
             <include name="**/*.properties"/>
         </classes>
         <!--lib dir="${lib}">
            <include name="**/*.*"/>
         </lib-->
         <webinf dir="${dirs.base}/WEB-INF/">
             <include name="**/*.*"/>
         </webinf>
           <fileset dir="${dirs.base}" includes="resources/**/*.*"/>
           <fileset dir="${dirs.base}" includes="web/**/*.*"/>
           <fileset dir="${dirs.base}" includes="javadoc/**/*.*"/>
           <!--
           <fileset dir="${dirs.base}" includes="test_web/**/*.*"/>
           <fileset dir="${dirs.base}" includes="src/**/*.*"/>
           -->
       </war>
     </target>

    <target name="generateEAR" depends="generateJAR, generateWAR">
        <ear earfile="${build.dir}/${ear.filename}"
appxml="${deploymentdescription}/ear/application.xml">
             <fileset dir="${build.dir}"
includes="${jar.filename},${war.filename}"/>
        </ear>
    </target>

    <!-- Copiamos el ear al directorio de despliegue -->
    <target name="moveEar" depends="init">
        <copy file="${build.dir}/${ear.filename}" todir="${deploy.dir}"/>
    </target>

    <target name="cleanServer" depends="init">
        <delete file="${deploy.dir}/${ear.filename}"/>
    </target>

    <!-- ================================================================== -->
    <!-- Cleans up generated stuff                                          -->
    <!-- NO FUNCIONA BIEN================================================== -->
    <target name="clean" depends="init">
        <delete dir="${ear.filename}"/>
        <delete dir="${war.filename}"/>
        <delete dir="${jar.filename}"/>
        <delete dir="${build.classes.dir}"/>
        <delete dir="${build.dir}"/>
    </target>

    <!-- Main target -->
    <target name="all" depends="cleanServer,clean,generateEAR,moveEar"/>

</project>
```

3. setup.xml. Instalación y Desinstalación.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="antprettybuild.xsl"?>

<project basedir="." default="init" name="Setup">

    <!-- ================================================================== -->
    <!-- ====  Properties declaration                              ===== -->
    <!-- ================================================================== -->
    <property file="setupUser.properties"/>

    <property name="sql.dir" value="./DataBase1"/>
    <property name="data.sql.dir" value="/Data"/>
    <property name="datamodel.sql.dir" value="/DataModel"/>

    <property name="create_users.id" value="CreateUsers"/>
    <property name="create_datamodel.id" value="CDM"/>
    <property name="drop_datamodel.id" value="DDM"/>
    <property name="prepare_datamodel.id" value="PDM"/>
    <property name="add_foreningkeys.id" value="AFK"/>
    <property name="insert_appdata.id" value="IAD"/>
    <property name="insert_intdata.id" value="IID"/>


    <property name="buildscript.class"
        value="edu.ucm.sip.commons.utils.SQLScriptBuilder"/>
    <property name="util.classes" value="./build/classes"/>
    <property name="libs.dir" value="./WEB-INF/lib"/>

    <property name="jdbc.driver" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="jdbc.url" value="jdbc:oracle:thin:@XXXX:1521:XXX"/>
    <property name="jdbc.user" value="xx"/>
    <property name="jdbc.password" value="xx"/>
    <!-- JDBC driver for Oracle8i -->
    <property name="jdbc.driver.jar" value="classes12.jar"/>
    <!-- JDBC driver for Oracle9i and jdk1.4-->
    <!--property name="jdbc.driver.jar" value="ojdbc14.jar"/-->
    <!-- JDBC driver for Oracle9i and jdk1.3-->
    <!--property name="jdbc.driver.jar" value="classes12.zip"/-->


    <target name="init">
        <tstamp/>
    </target>

    <target name="prepare" depends="init">
        <echo message="SQL dir: ${sql.dir}"/>
        <echo message="Util class: ${util.classes} -> ${buildscript.class}"/>
        <echo message="DATABASE: ${jdbc.url}"/>
        <echo message="USER: ${jdbc.user}"/>
        <echo message="PASSWORD: ${jdbc.password}"/>
    </target>

    <!-- ================================================================== -->
      <!-- ====   Create Users                                        ===== --
>
    <!-- ================================================================== -->
      <target name="create_users" depends="prepare">
        <echo message="Start script SQL: Create Users"/>
             <sql driver="${jdbc.driver}" url="${jdbc.url}"
            userid="${jdbc.user}" password="${jdbc.password}"
```

```
            delimiter=";" print="true" onerror="continue">
            <classpath>
                <pathelement location="${jdbc.driver.jar}"/>

            </classpath>
                    <fileset dir="${sql.dir}">
                           <include name="${create_users.id}.sql"/>
                    </fileset>
             </sql>
      </target>

    <!-- ================================================================== -->
      <!-- ====   Create Tables of DataModel                          ===== --
>
    <!-- ================================================================== -->
      <target name="build_create_datamodel" depends="prepare">
            <echo message="Building script SQL: Create Data Model"/>
       <java classname="${buildscript.class}" dir="."
            fork="yes">

                    <classpath>
                           <pathelement path="${util.classes.dir}"/>
                           <pathelement path="${libs.dir}/commons-logging-
1.0.3.jar"/>
                    </classpath>

                    <jvmarg value="-Xms128m"/>
                    <jvmarg value="-Xmx128m"/>
                    <jvmarg value="-Duser.region=EN"/>
                    <jvmarg value="-Duser.language=en"/>

            <arg value="${sql.dir}${datamodel.sql.dir}"/>
            <arg value="${create_datamodel.id}"/>
            <arg value="${sql.dir}/${create_datamodel.id}.sql"/>
            <arg value="clean"/>
            <arg value="v"/>
             </java>
      </target>

      <target name="create_datamodel" depends="prepare">
       <echo message="Start script SQL: Create Data Model"/>
             <sql driver="${jdbc.driver}" url="${jdbc.url}"
            userid="${jdbc.user}" password="${jdbc.password}"
            delimiter=";" print="true" onerror="continue">
            <classpath>
                <pathelement location="${jdbc.driver.jar}"/>
            </classpath>
                    <fileset dir="${sql.dir}">
                           <include name="${create_datamodel.id}.sql"/>
                    </fileset>
             </sql>
      </target>

    <!-- ================================================================== -->
      <!-- ====   Drop Tables of DataModel                            ===== --
>
    <!-- ================================================================== -->
      <target name="build_drop_datamodel" depends="prepare">
            <echo message="Building script SQL: Drop Data Model"/>
       <java classname="${buildscript.class}" dir="."
            fork="yes">

                    <classpath>
```

```
                        <pathelement path="${util.classes.dir}"/>
                        <pathelement path="${libs.dir}/commons-logging-
1.0.3.jar"/>
                    </classpath>

                    <jvmarg value="-Xms128m"/>
                    <jvmarg value="-Xmx128m"/>
                    <jvmarg value="-Duser.region=EN"/>
                    <jvmarg value="-Duser.language=en"/>

            <arg value="${sql.dir}${datamodel.sql.dir}"/>
            <arg value="${drop_datamodel.id}"/>
            <arg value="${sql.dir}/${drop_datamodel.id}.sql"/>
            <arg value="clean"/>
            <arg value="v"/>
             </java>
      </target>

      <target name="drop_datamodel" depends="prepare">
        <echo message="Start script SQL: Drop Data Model"/>
             <sql driver="${jdbc.driver}" url="${jdbc.url}"
            userid="${jdbc.user}" password="${jdbc.password}"
            delimiter=";" print="true" onerror="continue">
            <classpath>
                <pathelement location="${jdbc.driver.jar}"/>
            </classpath>
                    <fileset dir="${sql.dir}">
                        <include name="${drop_datamodel.id}.sql"/>
                    </fileset>
             </sql>
      </target>

    <!-- ================================================================== -->
      <!-- ====   Prepare Data Model                                    ===== --
>
    <!-- ================================================================== -->
      <target name="build_prepare_datamodel" depends="prepare,create_datamodel">
             <echo message="Building script SQL: Prepare Data Model"/>
        <java classname="${buildscript.class}" dir="."
            fork="yes">

                    <classpath>
                        <pathelement path="${util.classes.dir}"/>
                        <pathelement path="${libs.dir}/commons-logging-
1.0.3.jar"/>
                    </classpath>

                    <jvmarg value="-Xms128m"/>
                    <jvmarg value="-Xmx128m"/>
                    <jvmarg value="-Duser.region=EN"/>
                    <jvmarg value="-Duser.language=en"/>

            <arg value="${sql.dir}"/>
            <arg value="${prepare_datamodel.id}"/>
            <arg value="${sql.dir}/${prepare_datamodel.id}.sql"/>
            <arg value="clean"/>
            <arg value="v"/>
             </java>
      </target>

      <target name="prepare_datamodel" depends="prepare">
        <echo message="Start script SQL: Prepare Data Model"/>
             <sql driver="${jdbc.driver}" url="${jdbc.url}"
```

```
            userid="${jdbc.user}" password="${jdbc.password}"
            delimiter=";" print="true" onerror="continue">
            <classpath>
                <pathelement location="${jdbc.driver.jar}"/>
            </classpath>
                    <fileset dir="${sql.dir}">
                           <include name="${prepare_datamodel.id}.sql"/>
                    </fileset>
             </sql>
      </target>

    <!-- ================================================================== -->
      <!-- ====   Add Foreings Keys                                       ===== --
>
    <!-- ================================================================== -->
      <target name="build_add_foreingkeys" depends="prepare">
             <echo message="Building script SQL: Add Foreing Keys"/>
        <java classname="${buildscript.class}" dir="."
            fork="yes">

                    <classpath>
                           <pathelement path="${util.classes.dir}"/>
                           <pathelement path="${libs.dir}/commons-logging-
1.0.3.jar"/>
                    </classpath>

                    <jvmarg value="-Xms128m"/>
                    <jvmarg value="-Xmx128m"/>
                    <jvmarg value="-Duser.region=EN"/>
                    <jvmarg value="-Duser.language=en"/>

            <arg value="${sql.dir}${datamodel.sql.dir}"/>
            <arg value="${add_foreningkeys.id}"/>
            <arg value="${sql.dir}/${add_foreningkeys.id}.sql"/>
            <arg value="clean"/>
            <arg value="v"/>
             </java>
      </target>

      <target name="add_foreingkeys" depends="prepare">
        <echo message="Start script SQL: Add Foreing Keys"/>
             <sql driver="${jdbc.driver}" url="${jdbc.url}"
            userid="${jdbc.user}" password="${jdbc.password}"
            delimiter=";" print="true" onerror="continue">
            <classpath>
                <pathelement location="${jdbc.driver.jar}"/>
            </classpath>
                    <fileset dir="${sql.dir}">
                           <include name="${add_foreningkeys.id}.sql"/>
                    </fileset>
             </sql>
      </target>

    <!-- ================================================================== -->
      <!-- ====   Insert Application Data                                 ===== --
>
    <!-- ================================================================== -->
      <target name="build_insert_appdata" depends="prepare">
             <echo message="Building script SQL: Insert Application Data"/>
        <java classname="${buildscript.class}" dir="."
            fork="yes">

                    <classpath>
```

```
                        <pathelement path="${util.classes.dir}"/>
                        <pathelement path="${libs.dir}/commons-logging-
1.0.3.jar"/>
                    </classpath>

                    <jvmarg value="-Xms128m"/>
                    <jvmarg value="-Xmx128m"/>
                    <jvmarg value="-Duser.region=EN"/>
                    <jvmarg value="-Duser.language=en"/>

            <arg value="${sql.dir}${data.sql.dir}"/>
            <arg value="${insert_appdata.id}"/>
            <arg value="${sql.dir}/${insert_appdata.id}.sql"/>
            <arg value="clean"/>
            <arg value="v"/>
             </java>
      </target>

      <target name="insert_appdata" depends="prepare">
        <echo message="Start script SQL: Insert Application Data"/>
             <sql driver="${jdbc.driver}" url="${jdbc.url}"
            userid="${jdbc.user}" password="${jdbc.password}"
            delimiter=";" print="true" onerror="continue">
            <classpath>
                <pathelement location="${jdbc.driver.jar}"/>
            </classpath>
                    <fileset dir="${sql.dir}">
                        <include name="${insert_appdata.id}.sql"/>
                    </fileset>
             </sql>
      </target>

    <!-- ================================================================== -->
      <!-- ====   Insert Application Data                                 ===== --
>
    <!-- ================================================================== -->
      <target name="build_insert_intdata" depends="prepare">
             <echo message="Building script SQL: Insert Intranet Data"/>
        <java classname="${buildscript.class}" dir="."
            fork="yes">

                    <classpath>
                        <pathelement path="${util.classes.dir}"/>
                        <pathelement path="${libs.dir}/commons-logging-
1.0.3.jar"/>
                    </classpath>

                    <jvmarg value="-Xms128m"/>
                    <jvmarg value="-Xmx128m"/>
                    <jvmarg value="-Duser.region=EN"/>
                    <jvmarg value="-Duser.language=en"/>

            <arg value="${sql.dir}${data.sql.dir}"/>
            <arg value="${insert_intdata.id}"/>
            <arg value="${sql.dir}/${insert_intdata.id}.sql"/>
            <arg value="clean"/>
            <arg value="v"/>
             </java>
      </target>

      <target name="insert_intdata" depends="prepare">
        <echo message="Start script SQL: Insert Intranet Data"/>
             <sql driver="${jdbc.driver}" url="${jdbc.url}"
```

```
            userid="${jdbc.user}" password="${jdbc.password}"
            delimiter=";" print="true" onerror="continue">
            <classpath>
                <pathelement location="${jdbc.driver.jar}"/>
            </classpath>
                    <fileset dir="${sql.dir}">
                           <include name="${insert_intdata.id}.sql"/>
                    </fileset>
             </sql>
      </target>

    <target name="defaultdata"
depends="init,prepare,build_insert_appdata,insert_appdata,build_insert_intdata,in
sert_intdata"/>
    <target name="install"
depends="init,prepare,create_users,build_create_datamodel,create_datamodel,prepar
e_datamodel,build_add_foreingkeys,add_foreingkeys,defaultdata"/>
    <!--target name="clean"
depends="init,prepare,truncate_appdata,truncate_intdata"/-->
    <!--target name="unisntall"
depends="init,prepare,build_remove_foreingkeys,remove_foreingkeys,build_drop_data
model,drop_datamodel,drop_users"/-->
</project>
```